

A person is running on a sandy beach towards the left. The ocean waves are breaking in the background under a clear blue sky. The person is wearing a red top and dark pants. Their reflection is visible in the wet sand.

Mikko Välimäki

The Rise of Open Source Licensing

A Challenge to the Use of Intellectual Property in the Software Industry

PUBLISHED BY TURRE PUBLISHING, A DIVISION OF TURRE LEGAL LTD
Aleksanterinkatu 17, 6th floor, Helsinki, FI 00100, Finland, <http://pnb.turre.com/>

Copyright © 2005 Mikko Valmälä
First Edition. Some Rights Reserved



This book is licensed under the terms of Creative Commons Attribution-NonCommercial-NoDerivs 2.0 license available from <http://www.creativecommons.org/>. Accordingly, you are free to copy, distribute, display, and perform the work under the following conditions: (1) you must give the original author credit, (2) you may not use this work for commercial purposes, and (3) you may not alter, transform, or build upon this work.

ISBN 952-91-8769-6 (printed)
952-91-8779-3 (PDF)

Printed in the Helsinki University Printing House

Mikko Valimäki

THE RISE OF OPEN SOURCE LICENSING

**A CHALLENGE TO THE USE OF INTELLECTUAL PROPERTY IN THE
SOFTWARE INDUSTRY**

FOREWORD

This book is the result of my PhD studies at the Helsinki University of Technology. I started working towards a doctoral degree right after graduation from the University of Helsinki in 1999. I was supposed to write a thesis in law. Now I need to apologize my then-supervisors professor Niklas Bruun and docent Pekka Timonen not to complete the thesis at the law faculty in four years as was once planned. What happened was that I met my future academic mentor, professor Jukka Kemppinen, who had just started his professorship at the Helsinki University of Technology. He convinced me to change my plans and my university in the late 1999.

The actual theme of this thesis started to emerge during my year at UC Berkeley from 2000 to 2001. At that time I was working with Olli Pitkänen and we were supposed to study digital rights management. But I went on and spotted open source. I was lucky to participate at some of the first business and technology conferences ever that were arranged on the topic in California. I concluded that this is the area I have the best knowledge of and, besides, it doesn't seem to be a fad that disappears in the next two years. So why not write about it?

The main creative writing periods of this book were in October 2003 in Berkeley libraries and cafes, and August 2004 in the Starbucks of Santiago de Chile. In addition, there were also those numerable nights when I completed separate articles, which form considerable subparts of this thesis. I finished the work by completing all the open and missing parts under the supervision of professor Juha Laine.

Dissertation examiners professor Jukka Heikkilä and Dr. Ilkka Rahnasto made a number of substantial comments to a draft version of this book. I have taken most of them into account. Professor Thomas Riis from Copenhagen Business School kindly accepted the invitation to act as the academic opponent.

My understanding of open source has greatly benefited from discussions with those who practice software business. As the public interest in open source has grown, I have found myself lecturing and consulting open source licensing to different organization from Finnish software companies to Inter-

American Development Bank. Special mention goes to Antti Halonen who introduced me to MySQL before there was a company for that particular project. Márten Mickos, MySQL's CEO since 2001, has also been of help by giving constructive feedback and kindly sharing his connections.

Another bunch of special thanks go to my research colleagues Ville Oksanen and Herkko Hietanen. In addition to several co-authored research papers, the founding of Electronic Frontier Finland in 2001 has definitely sharpened my argumentation and overall writing skills. Through the association, I have had the opportunity to participate into the public policy discussion on copyright and patents from the inside.

I want to also thank Olga ja Kaarle Oskari Laitisen Säätiö, Jenny and Antti Wihuri Foundation, Helsingin Sanomien 100-vuotissäätiö, Soneran tutkimussäätiö, Ella and Georg Ehrnrooth foundation and the Research Foundation of Helsinki University of Technology for their grants supporting my research work when that support was most needed.

Finally, thanks to my family, friends and colleagues not especially mentioned. There are just too many people I've met at universities, conferences, business meetings and bars all around the world who have given their support and contribution in one way or other to this project. It makes me no sense to list you all.

"Meet the new boss – same as the old boss."

Lauttasaari, Helsinki, 30th March 2005

Mikko Valimäki

* Final verse from The Who song Won't Get Fooled Again (1971). Lyrics by Pete Townshend

TABLE OF CONTENTS

FOREWORD	III
TABLE OF CONTENTS	V
ABBREVIATIONS	IX
1 INTRODUCTION	1
1.1 PROBLEM	1
1.2 TERMINOLOGY, PERSPECTIVE AND LIMITATIONS	3
1.3 METHOD	5
1.3.1 Rationale for Different Methods Used	5
1.3.2 Continuing Patterns in Business History	6
1.3.3 An Economic Perspective	7
1.3.4 Comparative Law and Social Norms	8
1.4 ACADEMIC CONTEXT AND SOURCES	10
1.5 OVERVIEW OF THE STUDY	11
2 FROM PROPRIETARY TO OPEN: EVOLVING LICENSING MODELS IN SOFTWARE INDUSTRY	13
2.1 SOFTWARE INDUSTRY	13
2.1.1 A Short Historical Overview	13
2.1.2 Market Size and Regions	15
2.1.3 Emergence of Open Source	16
2.1.4 Open Source and Software Business Models	19
2.2 PROPRIETARY LICENSING	21
2.2.1 IBM's Unbundling Decision and Corporate Licensing	21
2.2.2 Mass Markets Licensing and Shareware	24
2.2.3 Proprietary Licensing Today	26
2.3 FREE SOFTWARE AND OPEN SOURCE LICENSING	30
2.3.1 BSD License and Unix Copyrights	30
2.3.2 GNU General Public License, Linux and SCO	33
2.3.3 Open Source Enters Vocabulary	36
2.4 SOCIAL AND POLICY DIMENSIONS OF OPEN SOURCE	40
2.4.1 Open Source and Individual Empowerment	40
2.4.2 Community and Its Coups	42
2.4.3 Ethical or Technical Goals?	44
2.4.4 Influencing Political Institutions	45
2.4.5 Practical Public Policy Initiatives	46
2.5 CONCLUSION: EXPLAINING THE INCREASING ROLE OF OPENNESS	48

3 ECONOMIC PRINCIPLES OF SOFTWARE PRODUCTS	50
3.1 ECONOMIC CHARACTERIZATION OF SOFTWARE PRODUCTS	50
3.1.1 A Network Economics Approach	50
3.1.2 Software as an Economic Good	51
3.1.3 Components and Systems	54
3.1.4 Path Dependence, Lock-In and Network Effects	56
3.2 ECONOMICS OF SOFTWARE COPYRIGHT	58
3.2.1 Motivation of Developers	58
3.2.2 Investors and Incentives	59
3.2.3 Costs of Copying	61
3.2.4 Optimal Limits of Copyright	62
3.2.5 Compensation Mechanisms	65
3.2.6 Is Software Copyright Inefficient?	67
3.3 ECONOMICS OF SOFTWARE INNOVATION AND PATENTS	69
3.3.1 Innovation in the Software Industry	69
3.3.2 Difficult Relationship Between Innovation and Patents	71
3.3.3 Patents as Strategic Assets	73
3.3.4 Different Means to Appropriable Innovation	74
3.3.5 An Open Innovation Model	75
3.4 COMPETITION POLICY AND THE LIMITS OF EXCLUSIVE RIGHTS	78
3.5 SUMMARY, ECONOMIC RATIONALE OF OPEN LICENSING	80
4 INTELLECTUAL PROPERTY AND ITS DISCONTENTS	82
4.1 CHALLENGE OF SOFTWARE PROTECTION	82
4.1.1 Early Discussion and Practice	82
4.1.2 WIPO's Proposal	85
4.2 COPYRIGHT AND ITS LIMITS	86
4.2.1 Software Enters Copyright Law	86
4.2.2 Interoperability Debate	87
4.2.3 Current Extent of Software Copyright	91
4.3 THE RETURN OF PATENTS	94
4.3.1 United States Leads	94
4.3.2 Europe Follows	95
4.3.3 International Policy	98
4.3.4 Current Extent of Software Patents	99
4.4 TECHNICAL PROTECTION	101
4.4.1 Early Copy Protection Systems	101
4.4.2 Anti-Circumvention Legislation	102
4.4.3 Is Technical Protection Effective?	103
4.4.4 The Promise of Trusted Systems	104
4.5 ARE INTELLECTUAL PROPERTY LAWS OUT OF BALANCE?	105
4.5.1 Balancing Principle	105

4.5.2 Expansion Trend	106
4.5.3 Open Source as a Balancing Force?	109
4.6 CONCLUDING REMARKS: AN OPEN PERSPECTIVE ON INTELLECTUAL PROPERTY	111
5 OPEN SOURCE LICENSES AS ALTERNATIVE GOVERNANCE MECHANISMS	113
5.1 BARGAINING IN THE SHADOW OF INTELLECTUAL PROPERTY LAW	113
5.1.1 What Makes a License 'Open Source'?	113
5.1.2 What Is Not Required?	114
5.1.3 Enforcing an Open Source Bargain	116
5.1.4 Licenses Categorized	117
5.1.5 Popularity of Open Source Licenses	121
5.1.6 A Framework for License Analysis	123
5.2 GNU GPL AND STRONG RECIPROCITY	124
5.2.1 Derivative Works in Copyright Law	124
5.2.2 Derivative Works and GPL	130
5.2.3 Patents and GPL	139
5.2.4 GPL and License Compatibility	140
5.2.5 Other Licenses with Strong Reciprocity	142
5.3 GNU LGPL AND STANDARD RECIPROCITY	146
5.3.1 LGPL Functionality	146
5.3.2 Other Licenses with Standard Reciprocity	148
5.4 BSD AND PERMISSIVE LICENSES	151
5.4.1 BSD Functionality	151
5.4.2 Other Permissive Licenses	152
5.5 EXCLUSION: CREATIVE COMMONS OPEN CONTENT LICENSES	154
5.5.1 Background	154
5.5.2 Creative Commons Functionality	155
5.5.3 Risk Allocation and Warranties	158
5.5.4 Internationalization and Formalities	159
5.5.5 Concluding Remarks	161
5.6 SUMMARY: COMPETITION BETWEEN EVOLVING LICENSING STANDARDS	161
6 DEFENSE WITH OPEN SOURCE: INFRINGEMENT RISK MANAGEMENT AND PATENTS	164
6.1 HOW TO MANAGE IPR INFRINGEMENT RISKS?	164
6.1.1 Background	164
6.1.2 Nature of Third Party IPR Infringements	166
6.1.3 Alternatives to Manage Risks	169
6.1.4 Actual Management Practices	174
6.1.5 Concluding Remarks	178

6.2 PATENTING PROBLEM AND POSSIBLE POLICY SOLUTIONS	179
6.2.1 Background	179
6.2.2 Open Source Licenses and Infringement Risk	180
6.2.3 Development Process from Patenting Perspective	182
6.2.4 Policy Debate on Open Source and Patents	183
6.2.5 Liability Exceptions for Open Source?	185
6.3 CONCLUSION: IPR LAWS CAN BE TUNED	186
7 OFFENSE WITH OPEN SOURCE: CASE STUDIES ON LICENSING	188
7.1 LICENSING OPEN SOURCE FOR PROFIT	188
7.1.1 Product Pricing Possibilities	188
7.1.2 Problem of Development Control	189
7.2 CASE STUDY 1: FREE LICENSES AND OPERATING SYSTEM SOFTWARE	192
7.2.1 Introduction	192
7.2.2 Market Overview	194
7.2.3 Study Framework	195
7.2.4 Microsoft Windows	196
7.2.5 Apple OS X	200
7.2.6 GNU/Linux Distributions	202
7.2.7 Concluding Remarks	204
7.3 CASE STUDY 2: DUAL LICENSING AND EMBEDDED SOFTWARE	206
7.3.1 How Dual Licensing Works?	206
7.3.2 Study Framework	208
7.3.3 Sleepcat Software Inc.	209
7.3.4 MySQL AB	211
7.3.5 TrollTech AS	212
7.3.6 When Does Dual Licensing Make Sense?	214
7.4 CONCLUDING REMARKS	216
8 CONCLUSIONS	218
8.1 THE RISE OF OPEN SOURCE	218
8.2 IMPACT ON LICENSING PRACTICES	219
8.3 IMPACT ON INTELLECTUAL PROPERTY MANAGEMENT	221
8.4 IMPACT ON COMMERCIAL REGULATION AND LEGAL STUDY	222
FIGURES AND TABLES	225
REFERENCES	227
ARTICLES, BOOKS AND REPORTS	227
NEWS, INTERVIEWS AND ONLINE SOURCES	237
COURT CASES, OFFICIAL DOCUMENTS AND LICENSES	245
INDEX	249

ABBREVIATIONS

BSA	Business Software Alliance
BSD	Berkeley Software Distribution
CC	Creative Commons
CPL	Common Public License
CP/M	Control Program for Microcomputers
EPO	European Patent Office
EU	European Union
FLOSS	Free/Libre and Open Source Software
FSF	Free Software Foundation
GNU	GNU's Not Unix [*]
GPL	GNU General Public License
IP	Intellectual Property
IPR	Intellectual Property Rights
IT	Information Technology
LGPL	GNU Lesser General Public License
MIT	Massachusetts Institute of Technology
MPL	Mozilla Public License
MS-DOS	Microsoft Disk Operating System
OSD	Open Source Definition
OSI	Open Source Initiative
OSL	Open Software License
OSS	Open Source Software
PC	Personal Computer
TC	Trusted Computing
US	United States
USPTO	United States Patents and Trademark Office
W3C	World Wide Web Consortium
WIPO	World Intellectual Property Organisation
WTO	World Trade Organisation

^{*} GNU is a recursive acronym

1 INTRODUCTION

1.1 Problem

MOUNTAIN VIEW, Calif. (February 23, 1998) - Netscape Communications Corporation (NASDAQ: NSCP) today announced the creation of mozilla.org, a dedicated team within Netscape with an associated Web site that will promote, foster and guide open dialog and development of Netscape's client source code. "Netscape is the first major company to exploit the power of the open source strategy," said Eric S. Raymond, open-source developer and advocate. "Making their client software source code free to developers is a bold move that will do great things for their products."¹

Craig Mundie, Microsoft CTO, May 16, 2001: "When comparing the Commercial Software Model to the Open Source Software model, look carefully at the business model and licensing structures that form their foundations. This comparison leads to the conclusion that the commercial software model alone has the capacity for sustaining real economic growth. Intellectual capital has always been, and will remain, the core asset of the software industry, and almost every other industry. Preserving that capital—and investing in its constant renewal—benefits everyone."²

This book is a study on how open source has challenged the thinking and actual use of intellectual property in the software industry. The emergence of open source software and the rapid expansion of the Internet have brought new software licensing practices to mass markets. For a short time at least, new entrants have challenged incumbents in the expanding software markets with the help of innovative copyright licensing strategies and courageous anti-patent policies.

¹ Excerpt from Netscape's press release mentioning term "open source" for the first time. See Netscape (1998).

² This was the message when Microsoft tried to push their share source initiative to compete with open source. See Mundie (2001).

GNU/Linux operating system, Apache web server and MySQL database are perhaps the best-known examples of open source software. On their part, the biggest companies in the industry from IBM to Apple have adapted to the changing environment with different open source licensing and operation strategies. But not everyone is winning; Netscape lost its market share despite the “bold move” to open source strategy announced in 1998. Also many other initiatives have had hard times to convince that going open source can indeed be a viable business decision.

This draws us to the main questions of this study:

- Has open source changed licensing practices in the software industry from a historical perspective? (chapter 2)
- Do the economic theories on software, copyright, and innovation work with the principles of open source? (chapter 3)
- Does open source challenge the development of software copyright, patents and other intellectual property laws? (chapter 4)
- What are the most relevant open source licenses and how are they built on intellectual property law and economic theory? (chapter 5)
- How actual are patent and other intellectual property infringement risks in the use of open source and how the potential risks can be defensively managed at the social policy and individual firm level? (chapter 6)
- Are there interesting industry cases where open source licensing models have been used as competitive tools? (chapter 7)

The overall argument of the book is that open source licensing *has indeed changed the ways the software industry thinks of and actually uses intellectual property*. Almost all major software companies in the world have since 1998 started to adopt open source licensing models as part of their business. Open source code and free copying and distribution models have made inroads to the intellectual property licensing practices of industry heavyweights. It may well be that in some application areas – such as the basic Internet infrastructure software – there are no long-lasting mass markets in the future for closed source code software products with re-

stricting licensing terms. However, it is very difficult to assess how fundamental and deep the change has actually been. In many sectors of the software industry – such as custom software development and specific industry applications – open source is still a non-issue.

This book further argues that the implications of open source to the management of intellectual property are twofold. First, intellectual property infringement risks must be taken more seriously when open source software is used. This is because *open source increases the negative effects from the continuous expansion of intellectual property rights*. Second, “Internet-businesses” are finally breaking through into software markets. This means that *the value of intellectual property increases from sharing but also becomes more complex to appropriate*.

Finally, this book argues that open source can have relevant implications on intellectual property rights policy. First, *openness balances commercial regulation*. Open licensing systems have proved how potential drawbacks from overregulation can be fixed without state intervention. In this way, open source also emphasizes a *more material study of intellectual property rights*. When a substantial number of right holders in a given industry decide not to enforce their core intellectual property rights – relying on economic-rational arguments – the premises of the policy discussion can be seen in a new light. Why and how do companies do that? What does it mean to the intellectual property system as a whole? Whether one should hold to the government granted intellectual property rights to the fullest is again a relevant question for any software developer and public policy maker alike.

1.2 Terminology, Perspective and Limitations

The main objective throughout this book is *intellectual property rights* (IPRs). They can be defined as government-granted limited – both in term and scope – monopolies (or privileges) to govern certain uses of software. This limited-monopoly definition assumes implicitly that the rights can’t be omnipotent but must be balanced somehow. For practical matters, the discussion is limited to copyright (governing “works”) and patents (governing “innovations”). In more formal legal literature, terms “the copy-

right protection of computer programs” and “computer implemented innovations” are commonly used. This book uses however more general terms *software copyright* and *software patents* when we speak of copyright and patents as applied to computer programs. Such terminology is also more common in economics and other social sciences literature.

Software licenses are contractual documents, which define how copyright and patent rights are used. A *licensor* is typically a software developer (software company) who licenses more or less of these rights to *licensees*. A licensee can be either another developer or end-user. Term *open source* is defined as a set of software licenses, which follow certain criteria further defined in Open Source Definition. Thus, this book studies how software companies apply intellectual property laws with open source licenses. – With *proprietary* software and licenses, this book refers generally to everything else but open source. This should not be seen as a clear-cut categorization, however. There are many proprietary licenses, which fulfill some of the criteria of open source (e.g. the sharing of source code, no copyright or patent royalties) but not all.

Software industry refers in this book to companies, which offer *business software products* for server and desktop computers. Specifically, it refers to those parts of the industry where open source licenses are being used extensively. It is obviously not possible to assess licensing practices and implication within the industry as a whole. Thus, for example, specific issues with computer games and embedded systems are largely omitted.

There are two leading perspectives in this book:

- First is that of a *software developer* – typically a software company. Since licensing is essentially one of the operative functions of any software company, it is natural to discuss how open source impacts licensing practices. Further, the developer perspective often fits well with the views of small or medium sized companies, or other independent ventures, whose entire business somehow depends on the licensing decisions.
- Second is that of a *public policy maker*. The general implications from changing licensing practices are inherently public policy questions.

This study identifies implications to the regulation of copyright and patents as well as intellectual property management within software companies.

In short, this book sees intellectual property rights “in action”. The selected perspectives necessarily limit the focus to some extent. For example the descriptive discussions on the historical evolution and (static) legal concepts of software copyright are in the end only supportive to the main theses of the book.

1.3 Method

1.3.1 Rationale for Different Methods Used

Chapters 2-4 build a theory of software licensing from historical, economic and legal perspectives. In the second chapter, the discussion draws from business history explaining the role of open software licenses in the software industry. In the third chapter, the economic theories on networks, copyright, patents and innovation are described and the role of open source licensing models within the theory discussed. In the fourth chapter, the method is mainly legal history describing how the legal protection of software has been developed and interpreted up to present day.

Rationale for this three-tier approach (history, economics and law) is an assumption that to fully understanding the impact of open source licensing one needs to be aware of a number of factors that software developers consider for licensing decisions. In addition to purely economic arguments, there are significant legal ramifications that determine available licensing options. And above all this, there are ideological, philosophical, historical, technical and social facts that may be decisive in some contexts. Simply put, a one-eyed view to the phenomena of open source software licensing would be filled with half-truths and errors.

Still, one needs to ask: why do we discuss different approaches in one study. Why don't simply delineate the problem under study so that for example only the legal nature and legal analytics of the licenses would be studied? Again here, it is assumed that such a separate study would have

less practical and historical importance than the one that would mix the knowledge from different fields of study.

1.3.2 Continuing Patterns in Business History

A few remarks on the method of the historical analysis practiced in this book. In historical research, this book stresses the role of continuing social patterns and processes over unique events. For example, the breakthrough of open source in the late 1990s had its roots in the communities of early computer hobbyists of 1960s. With necessary environmental changes, such as the growth of the Internet as a communication platform and shift in enterprise computing from mainframes to cheap microcomputers, open source code and the ideas of sharing and communities returned and took over.

There are substantial similarities in our analysis to the triumph of open source to Chandler's approach to the business history of electronics and computer industries of the 1900s as well as Christensen's approach to the patterns of technological innovation.³ First-movers and new innovators have been repeatedly able to gain (temporary) market power in the software industry until the proponents of new technological breakthroughs and ideas have taken control. In this sense, open source can be seen as a revolutionary new way of doing and distributing software in the new environment.

However, open source is not only a technological paradigm. It is even more about the fundamental ways of how software is being socially developed and legally licensed. In these senses, the change open source has brought to software industry seems, at least for now, more fundamental. Technology arguably changes much quicker than social and legal norms. Therefore, more emphasis is given to the social and policy dimensions than in the more traditional studies of the industry history.⁴ For instance issues such as the emergence of development community norms and the

³ See Chandler (2001) and Christensen (1997)

⁴ Lazonick (2003) points out the need to study how social and other institutional factors initiate economic transformation in a given industry. His theory is based broadly on North's (1981) somewhat more abstract theory on institutional change and its effect on innovation and entrepreneurship

formation of a legal policy towards interoperability and software patents are covered in this book with more detail.

Finally, in describing historical patterns one must be selective. It is obviously not possible to give a perfectly detailed account of all the events, which combined lead to particular outcomes. There are many alternative and convincing ways to tell the history of open source within the context of software industry. The author has used here a kind of bottom-up approach: for instance the history of software licensing is seen from the perspective of individual developers and license authors. In the end, open source licensing can be accounted to the ideas of individual programmers and their previous licensing practices. Thus, the history of for example shareware licensing in the 1980s needs more account than corporate software licensing practices from that time.

1.3.3 An Economic Perspective

Economic analysis in this book can be best described to follow the so-called pin-factory approach. In short, the aim is to understand how the business and economics of open source licensing function in practice through interviews and hands-on observations both in the real life and Internet discussions archives. Then, the result is explained in the terms of economics.⁵

Economic concepts used in the academic discussion on networks, copyright, patents and innovations are introduced and their applicability in open source development discussed. An obvious challenge in this kind of a descriptive approach is that many of these concepts are used in normative policy discussion such as the regulation software patents. The fact is that many academics may have normative goals hidden in their argumentation. That is the main reason why the discussion on economic concepts is kept rather abstract.

Although this book discusses the economics of copyright and patents among others, it should be clarified that the aim is not to follow a traditional law and economics approach. We accept that laws are always more

⁵ See Borenstein et al (1998)

or less inefficient. Further, it is useful to strive towards more social efficiency through legal development only when the law is seriously out of balance and when it can be enforced with relatively moderate costs. In many cases there are alternative governance mechanisms to complement the development of formal legal institutions.⁶

Thus, the main idea in this book is to study how open source licensing privately balances “normal” inefficiencies in intellectual property regulation and how licensing arrangements have perhaps caused different sorts of inefficiencies in the software markets.⁷ Also, we are interested in the strategic and practical possibilities software developers have for reacting to the economic implications of the existing laws. In this analysis, the game theoretical models built in the economics literature of business strategy – including the economics of networks and innovation – typically help out in pointing out the relevant environmental features that affect individual decision making.⁸

1.3.4 Comparative Law and Social Norms

Although most open source licenses have been written according to the United States law, each one should be legally interpreted in the jurisdiction the license is actually used. However, it must be stressed that the actual use of the licenses is flexible and extraterritorial – independent of possibly differing national legal interpretations. Distributed development and free redistribution on the Internet do not stop at national borders or national laws. Therefore, this book has an international and comparative approach to legal analysis.⁹

⁶ See e.g. Dixit (2004), pp. 1–14, presenting an overview of the approach he calls “lawlessness and economics” or “economics in the shadow of the law”.

⁷ See also Epstein (1997), pp. 1173–1174, who notes pessimistically that the economic analysis of legal doctrines has been so thorough that there are significant risks of just repeating of what is already known. Also advancing the research with more rigorous economic analysis has implied that the results are harder to understand and thus less useful. Interestingly, Epstein goes on to suggest that economic studies of law should focus to the evolution of particular institutions and social arrangements “such as the evolution of telecommunications, public utilities and nonprofit organizations.”

⁸ Shapiro (1989).

⁹ For practical reasons, we limit the study mainly to the US and EU laws.

Following Mattei, we see the sources of law in a competitive setting.¹⁰ There is no specific law on the rights to software and litigation on open source licensing issues has been rare. Thus, the interpretation of licenses should start from the principles of copyright and other intellectual property laws and already developed case law on software licensing.

In addition, also the so-called community norms must be taken into account as a competitive normative source. They are manifested for example in the ethical guidelines of the development community and lists of frequently asked questions by license authors and community spokespersons.¹¹ From a narrow legal perspective, the community norms can be seen as secondary norms reflecting the objectives of the licenses. In practice, the factual community norms may affect the behavior of software developers and users even more than formal laws and contracts. Potential reasons are that the applicable legal rules on intellectual property rights and licensing are unclear, developer communities form rather close-knit social networks that voluntarily avoid legal disputes, and that the costs of law enforcement on the Internet are high.¹²

Linus Torvalds has said on the risk that Linux's GNU General Public License (GPL) would not be honored: "My fears are mitigated by reality. Somebody might do it for awhile, but it is the people who actually honor the copyright, who feed back their changes to the kernel and have it improved ... By contrast, people who don't honor the GPL will not be able to take advantage of the upgrades, and their customers will leave them. I hope."¹³

As noted, actual legal conflicts and continuing license violations have been considerably rare taking into account the popularity of open source. This has been credited mainly to proactive community self-control and ef-

¹⁰ Mattei (1997), pp. 104–105, makes a clear difference between legislation and legal rules. He argues that legal rules are composed of any "legal proposition that affects the solution of a legal problem" independent of its origin. Attempts to strictly classify and limit the sources of law may lead to unrealistic views of the legal rules.

¹¹ See e.g. Debian Free Software Guidelines and Himanen (2001).

¹² See Ellickson (1991), pp. 282–283. It must be noted that Ellickson studied the social norms of a cattle rancher community so his results can be used only analogically.

¹³ Torvalds and Diamond (2001), pp. 96–97.

fective mutual conflict resolution for example by stopping the distribution of violating source code, rewriting violating parts or buying a proprietary license.¹⁴ On these grounds, legal research in this book does not stop at conservative risk analysis in chapter six but continues to investigate the possibilities and business effects of the new innovative licensing techniques in chapter seven.

1.4 Academic Context and Sources

There has already been published many books on open source from technical, business and social science perspectives.¹⁵ Hundreds of academic papers from different angles are readily available.¹⁶ There are also at least three books specifically on open source licensing.¹⁷ Most of the book-length material published on licensing so far is however rather practical and sharply restricted to legal analysis. To contrast, the aim of this study is to offer a comprehensive and academic, yet historically balanced and practically useful insight into the whole range of the open source licensing phenomena.

While the aim can be criticized as broad, the book does have a focus. The main academic tradition where this book can be connected to is the *law and economics of intellectual property rights*. Historical and legal analyses ultimately support the task of explaining how the growing popularity of open source licensing affects (if it does) the industry practices of intellectual property rights management.

Specifically, this book aims to contribute:

¹⁴ See e.g. Shmatz and Castiaux (2000), pp. 33–34 and Moglen (2001b).

¹⁵ For a technical book see e.g. Forrel and Fitzgerald (2002), business book see e.g. Fink (2003), and social science book see e.g. Weber (2004).

¹⁶ For example "Open Source Research Community" hosted at opensource.mil.cdn posted 56 research papers (mainly economics and other social sciences) on their website during 2004. Westlaw gives for the database "Journals & Law Reviews Combined" a total of 53 articles mentioning "open source" in their title; 25 of those were published in 2004 – As an anecdote, scholar.google.com (including all sort of journal articles, conference proceedings, draft online papers etc.) gives a total of about 2400 hits for articles, which have "open source" in their title, in March 2005.

¹⁷ Rosen (2004) and St. Laurent (2004) can be described as practical "hint books" written by lawyers to developers while Metzger and Jaeger (2002) is a detailed legal analysis of open source licensing in the context of German law.

- A business history perspective on the emergence of open source licensing as a pattern combining previous software licensing and development practices
- More in-depth analysis of the key intellectual property issues in open source licenses and their impact to software business
- New real world case studies on intellectual property risk management and licensing practices in open source development

Main sources of the study are academic literature in the software industry history, economics of software and intellectual property law on software. In addition, relevant trade journals, magazines and online sources are used to provide details for actual industry practices and commentary on historical events. In the empirical parts of chapters six and seven the data has been collected from interviews, questionnaires and available market data and statistics.

Many studies on software industry start by stating that there is not much previous research since the industry is still relatively young.¹⁸ While there is much true in this statement, it also undermines the amount of research efforts done. Today it is not difficult to find recent research on software industry starting from the industry history. The same can be said about economics; the central concepts used in this study such as the economics of networks, copyright, patents and technological innovation have been under major study already from the 1960s. Also the legal issues regarding software protection aren't new; the roots of modern software copyright and patents discussion are also in the 1960s. And as noted, during the last few years, the academic literature of open source has taken off.

1.5 Overview of the Study

The second chapter of the book describes the growth, size and segmentation of the international software industry. We are especially interested in how software has been sold and how open source and free software li-

¹⁸ See e.g. Torrist (1998) and von Westrap (2003)

censing has emerged to challenge established industry practices. We also discuss the issues of licensing in the industry policy development.

The third chapter discusses the economic theories of networks, copyright, patents and innovation in the context of software products. All theories characterize different aspects of software products bearing some differences but also many similarities. Aim of this chapter is to build a coherent theory where open source software licensing models can be analyzed within a larger economic context.

The fourth chapter discusses the evolution of legal and technical protection of software. Some commentators have criticized that the continuous expansion of different overlapping intellectual property laws over software has implied that the law is now substantially out of balance. We end the chapter by reviewing the evidence and discussing the possibilities of private balancing of intellectual property laws through open source.

The fifth chapter discusses how different open source licenses have evolved in practice. Licenses are categorized and their functionality further analyzed. Open interpretation issues as well as implications to software business are identified. Finally, the chapter ends with a discussion on open content, on how the techniques of open source licenses have been adopted for use in other works of art than computer software.

Any use of open source in business environment includes legal risks. In this chapter we discuss how software patents and other intellectual property infringement risks can be managed at individual firm and social policy level. We start from the more general intellectual property rights infringement risk management alternatives in open source development. From there, the discussion is extended to the social problem of software patents especially in the European policy context.

Finally, the seventh chapter studies how open source licenses have been used offensively as a part of market-changing business strategy. First case study is about operating system software markets and how open source alternatives have changed the existing market structure during the recent years. The second one describes a specific open source licensing model called dual licensing and discusses how several start-up companies have benefited from using it.

2 FROM PROPRIETARY TO OPEN: EVOLVING LICENSING MODELS IN SOFTWARE INDUSTRY

This chapter describes the growth, size and segmentation of the international software industry. We are especially interested in how software has been sold and how open source and free software licensing has emerged to challenge established industry practices. We also discuss the issues of licensing in the industry policy development.

2.1 Software Industry

2.1.1 A Short Historical Overview

Software industry is an ambiguous concept. The industry is both young and characterized by rapid technological development. The industry has been shaped by numerous expansion times and technological paradigm shifts during its less than fifty year history. Thousands of companies have grown and disappeared. Today, only few companies that operated in the 1950s and 1960s have survived and most of the contemporary industry leaders were founded less than thirty years ago.

Campbell-Kelly offers a useful taxonomy of software industry from historical perspective. He identifies three major categories of software companies based on their operating model: software contractors, corporate software producers and mass-market software producers.

Category	Software contractors (1950s -)	Corporate software producers (1960s -)	Mass-market software producers (1970s -)
Companies	SDC (1956) CUC (1955) CSC (1959)	SAP (1972) CA (1976) Oracle (1977)	Microsoft (1975) MicroPro (1978) Lotus (1982)
Business	Projects	Product tailoring, services	License sales

Table 1. A historical taxonomy of the software industry¹⁹

¹⁹ Campbell-Kelly (2003), p. 9

In the first wave came software contractors. They established the software industry in the 1950s selling large-scale software projects to the United States government and largest corporations. In the 1960s the industry slowly shifted towards software products and expanded to Europe and other continents. After IBM, the industry's "natural" monopolist, unbundled software from hardware in 1969, the software product markets took off. Many new software products companies were founded and the industry expanded to serve a greater scale of users. Software services, training and support were major income sources to software producers at that time. What contractors and product companies had in common was that they both served mainly the corporate market, which was based on main-frame hardware.²⁰

Personal computer revolution was the next turning point in the industry. A new form of software business was mass-market software based from the beginning on copyright license sales and minimal after-sale services. This new part of the industry was in many ways disconnected from the corporate software markets. Mass-market software markets expanded rapidly in the "gold rush" of the late 1970s and early 1980s.²¹ In 1981 IBM introduced its PC, which was based on open architecture meaning that third party manufacturers were able to produce both software and hardware.²² PC became soon the dominant technology and helped the most successful of the new entrants to grow to the heights of the corporate software companies.

The focus of this study is on software producers. Since the late 1990s the boundaries between the corporate and mass-market software producers have started to some degree melt. Both the growth of Internet and the emergence of open source products have catalyzed a process of building the bridge between corporate and end-user markets. For example Microsoft, which has been a typical example of a mass-market software products company, and Oracle, an archetype of a corporate software company,

²⁰ Campbell-Kelly (2003), pp. 118-119, 161-162.

²¹ Campbell-Kelly (2003), p. 203.

²² According to Grindley (1995), p. 141, major reasons for open architecture were rush to markets, low-risk because of reliance in third party suppliers, and technical difficulties to protect the complex standard. IBM did try to make BIOS chip design proprietary but third party manufacturers soon circumvented it.

compete today in corporate databases and personal computer applications.²³

2.1.2 Market Size and Regions

It is difficult to estimate the size and importance of software products markets today. In many ways, software has become invisible. Software can be found in all kinds of products from cars to tennis rackets.²⁴ The figure below shows further how the software markets in the United States has been growing – save for a historical drop in 2002 – and is now valued at well over \$300 billion:

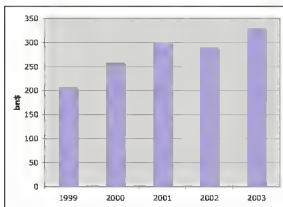


Figure 1 Total revenues from the top 500 US-based software companies.²⁵

²³ Campbell-Kelly, p. 11.

²⁴ Carr (2004) offers a critical view of the business impact of this commodification trend in the IT industry at large: it is becoming harder to get any competitive advantage from merely using certain kind of software if it is basically available to all.

²⁵ Collected from Software Magazine's "Software 500" annual surveys between 2000 and 2004. According to their methodology "Software revenue includes license fees, as well as revenue from product maintenance and support and professional services. Revenue from training, hardware sales, and peripherals are excluded." See also The Economist (2001) for comparison.

Most reliable estimates, such as European Information Technology Observatory, address around 30% of the markets to the United States and around the same to the Western Europe with practically all major software companies located in the US.²⁶ There are many reasons for United States dominance in the world software markets. Naturally, United States has the biggest domestic markets while for example in Europe local languages and other structural differences have made progress slower. Software development is also closely tied to hardware and the leading hardware manufacturers have been US companies. IBM's unbundling decision, which opened software markets in 1969 affected the US markets first and gave it a historical first-mover advantage as a nation. Industry hotspots such as Silicon Valley sprang up linking together entrepreneurs, university research, capital markets and other supportive services.²⁷ Software industry in the United States has also enjoyed strong public support since 1950s while in Europe the focus on national leaders specialized in hardware and electronics. Also the legal development in Europe has followed that of the United States in both intellectual property and competition law. Finally, according to Tornisr's study, US companies have been more product and R&D specific than their European counterparts.²⁸

Also this study reflects the historical US dominance in the software industry. The fact is that the biggest software companies are based in the United States and also licensing practices have been developed largely according the US legal and business traditions. Thus, if open source licensing and business models really challenge the software industry, the effects should be clearly visible in the United States markets.

2.1.3 Emergence of Open Source

It is even more difficult to estimate the size of "open source software industry". Pure open source companies are tiny and many of them are privately held. However, the popularity of open source is significant and practically all big IT companies have today open source products and serv-

²⁶ European Information Technology Observatory (2004)

²⁷ See e.g. Kenney (2000).

²⁸ Tornisr (1998), p. 155-156

ices available. The open source market has grown in side with the expansion of the Internet since open source products practically run the Internet.

Open source software has perhaps shaped most the markets of web server software, which has become a new market for software products between personal computers and corporate mainframes. A software combination known with acronym LAMP (Linux, Apache, MySQL and PHP/Perl) has been the first choice for many system integrators during the recent years, IBM and Oracle for example sell Linux and Apache-based server solutions; of course, Oracle may run their own database on top of it and also IBM has its own database products.

Figures 3 illustrates the popularity of Apache web server, which also indicates the popularity of open source software on web server markets:

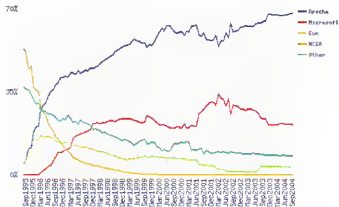


Figure 2. Market share of web servers²⁹

Pie in figure 4 describes the different segments of web server market in the early 2000s. It is possible to build a reliable web server using only open source components.

²⁹ See Netcraft (2004) for a continuously updated survey based on over 40 million server sites

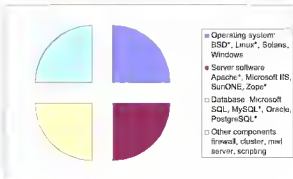


Figure 3. Main web server software components in the early 2000s.³⁰

The popularity of open source illustrates how standardization and commoditization in both hardware and software components have resulted in disruptive innovative activity. Major steps in hardware innovation have had implications to software innovation alike.³¹ The fast growth of networking equipment has implied rising demand to cheap, reliable and secure platform software where dominant PC platform Windows has not been suitable. Microsoft has been developing quickly its Windows NT technology on that market. On the other hand, expensive corporate mainframes with proprietary Unix platforms were not targeted to the low end of the market where the fastest expansion of the Internet has happened.

However, popularity does not directly imply shares of revenue. Open source is typically priced significantly lower compared to proprietary products. According to an IDC, Microsoft's share of all revenue generating server shipments in 2002 was 55% while Linux held only 23%.³² These two were the only server operating systems with a growing market share. IDC also estimated that the total market revenue was around \$18.6 billion.³³

³⁰ Products marked with star (*) are open source.

³¹ Torrisi (1998), p. 154

³² According to IDC (2003), "Only paid software shipments, whether included with hardware or shipped exclusive of hardware, are incorporated in the revenue shipment statistics."

³³ IDC (2003).

Also, open source has not been that successful in personal computer desktop software so far. The market shares have not been changed much. If one uses searches made on Google as an indicator, during June 2001 and June 2004, a steady 1% of all searches came from computers using Linux as the operating system. The market share of Mac OS has been around 3-4% while other non-Windows operating systems gaining another 4%. The rest of Google queries, that is over 90%, were made from computers running Microsoft Windows.³⁴ There is also reliable open source software available in major application software categories including office software (word processor, spreadsheet, and presentation), web browsing and email but it has proved difficult to get any relevant market share from the dominating Microsoft products.³⁵

2.1.4 Open Source and Software Business Models

Software business models used today can be distinguished from several perspectives. For example, depending on whether the software is sold as a product or service, structure of the sales channel, and income sources.³⁶ Table 2 identifies one possible classification used in the literature based on four rather generic models:

³⁴ Information collected from Google Zeitgeist (2004). Google is currently the most popular search engine on the Internet, which handled in early 2003 around 250 million queries per day according to OneStat (2003a) report. Google Zeitgeist's information on operating system market shares is only indicative. For example OneStat (2003b) reported that among web users Linux would have 0.5% share and Apple only 1.5% with over 97% going to Microsoft.

³⁵ OneStat (2004) reported that Microsoft Internet Explorer has over 93% dominance in web browsers with open source based Mozilla engine (Firefox, Mozilla and Netscape browsers) having a combined share of around 5%. Firefox is currently eating the market share of Explorer after the release of Firefox 1.0 in November 2004. This is the first time since Mozilla went open source in 1998 that it has taken any relevant market share back from Microsoft's Internet Explorer.

³⁶ See Rajala et al (2001).

	Software projects	Software publishing
Product focus	Customer project	Product family
Copyright	Licensed or transferred	Licensed with restrictions
Income	One-time project fees	License fees
	Software subscription	Commodity software
Product focus	Parametrized product	Core product
Copyright	Licensed with restrictions	Licensed with an open source license
Income	Service fees and application rents	Indirect from services, bundling, branding

Table 2 Generic software business models³⁷

Perhaps the most common way of software companies to do business is to sell *software projects*. In this model, a company sells its programming work as a service rather than the sole software. Project business is not much different from a taxi service where more cars running (programmers) mean more money to be charged. For example large IT enterprises such as IBM follow the project business model when they sell "system integration" services.³⁸

Next, the traditional model for software product business could be described as *software publishing*. In this model, the software is licensed as if it were sold as a physical product. Software publishing works in a somewhat similar way to print publishers who sell physical books commoditized from manuscripts. Microsoft has been a classic example here selling Windows operating system and Office applications.

The Internet as usage environment and distribution channel has enabled several new ways to do software business. *Software subscription* can be seen as a combination of the two traditional models. Sometimes called as application service providing, subscription is a more interactive way to sell software as an online product with add-on services tailored to the customer. Almost any software company selling software-based services on the Internet such as online marketplaces can be classified following the subscription model.

Finally, different *commodity software* business models have emerged. Here, a core product or a standardized component is available free of

³⁷ Table edited from Rapla et al. (2001).

³⁸ Of course, large companies such as IBM do software business under several models projects being just one of them.

charge or with nominal costs. Most popular open source software products such as Linux and Apache can be put into this category. By definition, all sales in the commodity model are based on indirect means that leverage the potentially large and dynamic user base. For example, add-on services, bundled products and branding are essential indirect revenue sources.

Of course, any classification of software business models can be criticized. One can for instance say with good grounds that there are *just product (licensing) and service business models*.³⁹ For example publishing and subscription can be characterized just as different licensing models within the general category of software product businesses. Table 2 above was based on this simple categorization.

Where does open source fit into the picture? Clearly, open source does not end in the commodity software business model as described in the early literature.⁴⁰ As we will see, many large IT companies such as IBM, HP and Novell selling software projects (services) use extensively open source software components. Open source also allows software subscription to a large extent. Most popular websites, be they commercial or not, run basically on open source. Finally, we can identify open source companies selling traditional licenses in addition to the free offering and thus using the publishing model. Start-ups such as MySQL, TrollTech and Sleepycat Software have pioneered with this approach.⁴¹ The result is that open source software can be combined into any popular software business model. It is not meaningful to speak of open source software business as if that would denote to a specific software business model.

2.2 Proprietary Licensing

2.2.1 IBM's Unbundling Decision and Corporate Licensing

Before there were any separate markets for software, computer hardware and software was sold together. Pugh traces this practice of bundling

³⁹ For example Cusumano (2004) builds his framework on software business on this distinction between (licensed) products and services

⁴⁰ Raymond (1997) was perhaps the first to analyze open source business models under the commodity model.

⁴¹ Their business model will be discussed in detail in chapter 7.3

back to Herman Hollerith who won the contract to tabulate US Census of 1890. Because his first customers did not know how the systems worked or did not want any responsibility, it was natural to rent the equipment and sell additional services. Hollerith's company later became IBM and the practice of bundling continued in the eras of Thomas J. Watson Sr. and Jr.⁴²

IBM essentially bundled software with hardware and services. They rented bundled software and hardware and sold additional support and maintenance. In a way, software was free and there were no extra charges to obtain IBM software. On the other hand, there were really no markets for the software on any other platform than IBM computers. For the first part of the century, IBM had a natural monopoly in the computer industry.

IBM announced the unbundling of its software from hardware in 1969 after series of internal studies. Unbundling was based on both competitive factors and antitrust fears.⁴³ It was claimed that IBM used its monopoly position against fair commercial practices. It offered free software and services according to customer needs and had therefore advantage in winning new customers. It was also claimed that IBM made it difficult to develop interoperable systems and cut prices to hinder competition.⁴⁴ Interestingly, after the unbundling decision hardware prices went down by nominal 3% while many customers hoped for as much as 25% hardware price reductions.⁴⁵

Right after the unbundling decision, IBM took into use several licensing models for selling software. They offered both one-time fees and site licenses. Watts S. Humphrey, then an IBM employee and involved in many of the unbundling decisions, later explained how they came to trust on a legally enforced licensing model instead of technical enforcement (copy protections etc):

"While we thought that cryptography might be technically feasible, particularly with special hardware assists, every approach we

⁴² Pugh (2002).

⁴³ A federal antitrust investigation against IBM's practices in the computer industry was filed in January 1969. The investigation wasn't dropped until 1982.

⁴⁴ Grad (2002), p. 64

⁴⁵ Humphrey (2002), p. 61.

could think of would have made it difficult for reputable customers to use our programs. Large businesses often needed backup copies, programs were frequently moved among machines, and IBM encouraged upgrading to larger systems. With cryptography, these activities would all require IBM permission. We felt that this would be impractical and inconvenient for users and expensive for IBM. We also concluded that any single-machine locks and keys, or special time-out and self-destruct programs, would be onerous to our best customers and not effective against clever thieves. Because we could not devise practical physical security measures, we had to rely on the inherent honesty of our customers. Our hope was that legal protection and criminal prosecution would limit the piracy problem."⁴⁶

While undoubtedly important, the significance of IBM's unbundling decision should not be overemphasized. Most industry professionals agree it was a necessary step leading to the birth of the software products industry in the 1970s.⁴⁷ However, it was just part of the story. According to Campbell-Kelly, the major factors leading to the emergence of the software products industry in the late 1960s were the increasing proliferation of computer uses, increasing software development costs, lack of programmers, and the introduction of IBM's standard platform.⁴⁸ On this background, software unbundling can be seen as a rational business decision in the evident evolution towards software markets.

Following IBM's policy change, software was sold as a licensed product from the late 1960s. Already early license agreements regulated software use in detail.⁴⁹ The legal basis for licenses was essentially trade secret and contract law although also intellectual property rights were sometimes mentioned (but not necessary claimed ownership upon).

⁴⁶ Humprey (2002), p. 60-61.

⁴⁷ Grad (2002), p. 71

⁴⁸ Campbell-Kelly (2003), p. 89

⁴⁹ For example Bigelow (1970) mentions that "...some contracts limit the number of central processors on which the program can run or the number of copies of the program that can be made". He also points out an interesting fact that "...ownership can become particularly sticky when the user improves the program, and wants to claim some rights in the improved version."

For example first Unix licenses from the early 1970s mentioned copyright, trade secrets and patents. The licenses were constructed from an assumption that Unix and Unix source code were under a trade secret and licensed under confidentiality. Further, the terms restricted copying as any copyright license – however, without explicitly mentioning that the software could be covered by copyright. The terms also stated that there is explicitly no patent or trademark license even if there would be patents or trademarks covering the software and that there is no warranty for third party copyright, patent or trade secret infringement.⁵⁰

2.2.2 Mass Markets Licensing and Shareware

Personal computer mass markets applied the publishing business model of the recording industry. Software packages were sold as copies of the original program. Popular books aimed at software developers explained that copyright was the main legal tool to protect software from the negative effects of unwanted copying. In effect, intellectual property protection grew in importance and licensing models became more and more restrictive banning e.g. backup copies and reverse engineering.

This wasn't the whole picture, though. Mass markets also saw the rise of shareware or "try-before-you-buy" licensing model in the early 1980s. The first successful shareware programs were PC applications but later also system tools and games were successfully distributed as shareware.⁵¹ In the early 1980s, it was basically possible to any competent individual to write competitive simple programs for such basic tasks as text processing. The only problem was that software publishing markets were still about to form and publishers always took their share from revenues.

Shareware became perhaps the first licensing model where software was distributed by end-users. Essentially, software was copyrighted but copying and distribution was allowed. However, to use the software for more

⁵⁰ See Unix License (1974)

⁵¹ See Ford (2000) on the early history of PC shareware software. At first, shareware was called freeware. Later, freeware became to mean software distributed in binary form for which the author didn't want any payment

than a certain time period or to receive additional features, the author typically required users to pay a license fee for registered or full-feature version. Also, source code didn't follow with the program and modification wasn't allowed.

Shareware programs were spread first through bulletin board systems and direct sharing among users. Later, mail order vendors appeared publishing lists of available shareware and charging little fees from disk production. This low-end of the whole software product markets was very intense. Nelson Ford, the founder of Public Software Library (PsL), one of the first shareware distributors, later explained:

"While PsL and the high-volume dealers dominated the shareware distribution market, during the late '80s, hundreds (if not thousands) of small shareware vendors sprang up. With no real computer knowledge or other expertise required, anyone with a few bucks could buy shareware disks from another vendor, print out a "catalog", and sell copies of those disks to others. Most of these "shareware vendors" sold at computer shows and flea markets."⁵²

The popularity of shareware slowly faded during the early 1990s on PC markets. Only few companies made significant profit before most of their software products became commoditized.⁵³ When for example Microsoft bought or developed some essential shareware tool into its operating system, the market for that shareware product eventually died. While new shareware programs were continuously introduced, the success stories became few and rare.⁵⁴

⁵² Ford (2000)

⁵³ See Takeyama (1994) for some estimated return rates in the shareware industry in the early 1990s. Only very few products returned to their developer any significant income; however, those few products may have been very profitable.

⁵⁴ Of course, there are many counterexamples. For example Paint Shop Pro, launched in the early 1990s, was able to gain significant market share among bitmap image processing software products.

2.2.3 Proprietary Licensing Today

License restrictions and pricing. Traditionally proprietary software companies have developed software in-house and used various kinds of end user license agreements that give licensees limited rights to use the software for specific purposes. The basic idea is to tie the license price with usage restrictions.

Table 3 below lists some typically used restrictions, which may be based on e.g. software itself, hardware environment, software users, or usage characteristics:

Criteria	Restriction type
Software	Copies (e.g. one copy, one site)
	Functionality (e.g. versioning)
Hardware	Configuration (e.g. number of processors)
	Computing power (e.g. transactions per second)
Users	Number of users (e.g. floating, fixed)
	Status of users (e.g. personal, educational)
Usage	Transactions (e.g. number of functionality used)
	Time (e.g. annual, perpetual)

Table 3 Typical proprietary license restrictions

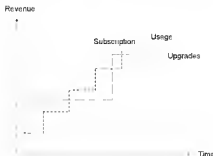


Figure 4 Illustration of licensing models from revenue generation perspective over time³⁵

³⁵ Messerschmitt and Seypeński (2003), p. 329

In the economic literature software licensing models have been discussed under such topics as price discrimination and versioning. Products can be differentiated and priced for example through release delay, quality discrimination, upgrades, renting, and bundling.⁵⁶ Theoretically, price discrimination maximizes the value from software use as each user pays according to his individual valuation. Intellectual property rights – giving software producers the exclusive rights for copying, distribution and modification – provide the necessary legal backup for such licensing models.

Licenses versus services. Licensing fees (royalties) form only a part of the total income of software companies. Even companies with licensable and customer-installable products typically engage in after-sale services and maintenance. It is therefore interesting to note that while different licensing models have been multiplied, average license prices have decreased in the long term.⁵⁷ At the same time, total revenues of software companies have increased from both product licensing and other sources. Thus, overall software usage must have increased very rapidly.

The following table further illustrates how the revenues of the world's largest software companies are shared between licenses/products and services/maintenance.

⁵⁶ See Shy (2001), p. 182-184, and Varian and Shapiro (1999).

⁵⁷ See Liebowitz and Margolis (2001), p. 154-157, noting especially that in those markets where Microsoft has participated, license prices have dropped significantly. Short-term picture may be more complex. Industry analyst firm Gartner recently warned that hardware based corporate licensing costs are about to increase at least 50% partly because of evident changes in computing architectures. See McCue (2004).

Company	Licenses / Products	Services / Maintenance	Revenue in millions	Software products
Adobe	98	2	1667	Graphics, publishing
Symantec	98	2	1870	Security tools
Microsoft (1)	94	6	31521	Operating systems, applications
Oracle	79	21	10156	Databases
CA (2)	69	31	3276	Applications, middleware
Siebel	36	63	1354	Enterprise applications
SAP (3)	31	69	9768	Enterprise applications
Novell (4)	26	74	1166	Operating systems, applications
IBM (5)	25	75	61307	Operating systems, applications etc.

(1) HPN and Home Entertainment divisions omitted

(2) Licenses include subscription services, which may combine some services

(3) Licenses include product maintenance, revenue calculated from exchange rate 1 EUR = 1.39 USD

(4) Linux subscriptions included in services

(5) Licenses include software product maintenance etc., services include large scale information technology services

Table 4. Software licenses and services revenue of some of the world's largest software product companies⁵⁸

It must be noted that this kind of calculations between the shares of licenses and services revenue are only indicative since company accounting standards are not fully comparable. However, the figures clearly show that some software companies are totally dependent on license sales and many companies make a substantial part of their revenue from licensing fees and royalties. As we will show later, the growing popularity of open source potentially undermines businesses models, which build on license fees and royalties.

However, licenses have also other than revenue generating functions. Particularly in open source licensing, the function of the license is not to generate direct royalties but add some other restrictions aiming for e.g. development cooperation, author attribution or even some form of ideology. Thus, software licenses can be seen in a larger context as part of the software's functionality and not just the price.⁵⁹

⁵⁸ Data collected from 2004 annual reports (form 10-K in the US – only SAP is an European company). Companies were selected based on the facts that they have major software products, over 1 billion in revenue, were featured in the top 50 of Software 500, and the share of licenses/product sales could be calculated from the annual reports. The last requirement was the most difficult to fill. Thus, for example Sun Microsystems and HP are not featured in the table.

⁵⁹ Interestingly, an OECD report on software measurement defines software as "computer programs, program descriptions and supporting materials for both systems and applications

Source code. Usually, source code is not shared in proprietary licenses and the software product is distributed only in object code (or intermediate object code as in Java) with additional restrictions on reverse engineering. The assumption is that source code contains valuable trade secret information, which should be protected from the eyes of the competitors. Copyright and patents do not give any protection towards e.g. the structures, ideas and logic described in source code.

Sometimes source code is however needed, especially if the software product is a development tool or a component, which needs integration with other components (embedded software). Licensees expect more adaptability and therefore source code or at least detailed interface descriptions must be made available.⁶⁰ Figure 5 below explains options for software distribution from source code perspective.

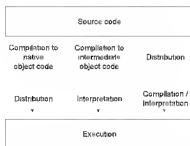


Figure 5 Three main ways to distribute software products from source code perspective.⁶¹

To conclude, it should be noted that also proprietary software is sometimes sold with source code. However, giving the source code of proprietary software to the user usually means higher costs of license enforce-

software. Licenses to use or reproduce software are not separated from the underlying software, and are thus included in this category." See OECD (2003), p. 11.

⁶⁰ See e.g. Chávez et al (1998), p. 49.

⁶¹ Edited from Messerschmitt and Szyperski (2003), p. 102.

ment; with source code at hands, the user has much more possibilities to use the software in ways not allowed by the licensor.

2.3 Free Software and Open Source Licensing

Next, we shortly review the history of free software and open source licensing. The ideas of liberal distribution terms with available source code were codified and then became popular with two major operating system projects BSD and GNU/Linux starting from the 1980s.⁶² Open Source Initiative later introduced the umbrella term “open source” to describe different types of free licenses in the late 1990s.

2.3.1 BSD License and Unix Copyrights

University background. In academic circles software had been for a long time developed with the principles of open source code and free distribution. Many universities chose to use AT&T’s Unix operating system. It was licensed from the beginning to educational institutions with full source code under a trade secret agreement.⁶³ Users were then encouraged to develop the system further – in fact, this was also a practical necessity, since AT&T did not really support the system. An evident implication of AT&T’s policy was that Unix became the basis for the first large-scale open collaboration development network.⁶⁴

A major variant of AT&T’s Unix became from the University of California at Berkeley. Most notable Berkeley-hacker Bill Joy, then a computer science graduate student, started to work on what eventually became Berkeley Software Distribution (BSD) in 1975. BSD soon became the academic Unix development platform. If users sent their hacks, patches and fixes to Berkeley, and they were accepted, the contributed code was added to the

⁶² As noted, BSD and GNU were not the very first sources of “liberal” licenses in the sense that already many shareware and public domain software was distributed with extremely liberal terms (if any) in the early 1980s. However, from historical perspective, BSD and GNU are the grandfathers of the open source licenses used today.

⁶³ AT&T’s Unix in fact didn’t have a copyright notice until 1984. See *USL v. BSDI* proceedings.

⁶⁴ Salas (1994) notes that AT&T’s approach to Unix was “no support, no bug fixes, and no credit”

BSD code base.⁶⁵ Much of the Unix development in fact happened in BSD, which was funded generously by Defense Advanced Research Projects Agency (DARPA) in the United States.⁶⁶

Bob Fabry, the head of Berkeley's Computer Science Research Group until 1983, described their motivation in an interview:⁶⁷

"I think the spirit in which we were putting this all together was much the spirit that was picked up later by the Free Software Foundation and the various people who were trying to build 'software for the people ... The idea is that there is no duplication cost for software, so it ought to be basically free, and we were all working together to try to produce this ideal system that we would all love to have, and love to be able to use ourselves."

To avoid the problems with possible copyright violations, license fees were paid to AT&T for any distribution of Unix variants. For example all BSD distributions included from the early 1980s a reference to AT&T's license. Nevertheless, the continuously rising license payments became soon burden for many.⁶⁸ Also companies, who used only parts of the code and built stand-alone networking products for the growing personal computer markets requested a separate version for their needs.

Finally, an independent creation from Berkeley called Networking Release 1 saw the daylight in June 1989. It was distributed under the first modern BSD license. Berkeley computer scientist Marshall Kirk McKusick later explained:⁶⁹

⁶⁵ The acceptance rate wasn't very high. Bill Joy has remembered in an interview that he didn't really get much contributions from outside. According to Kirk McKusick, who became in the 1980s BSD project lead, over 90% of contributions were rejected. See Leonard (2000a).

⁶⁶ Leonard (2000a), McKusick (1999).

⁶⁷ Leonard (2000a).

⁶⁸ During an anti-trust investigation, which had started in 1958, AT&T was obliged to license all but its telephone technology with reasonable terms to anyone. After the suit finally ended and AT&T was broken up in 1984, the Unix license fees started to rise from a nominal \$99 to thousands of dollars. See Leonard (2000), and The Economist (2004b).

⁶⁹ McKusick (1999).

"The BSD originated networking code and supporting utilities were released in June 1989 as Networking Release 1, the first freely-redistributable code from Berkeley. The licensing terms were liberal. A licensee could release the code modified or unmodified in source or binary form with no accounting or royalties to Berkeley. The only requirements were that the copyright notices in the source file be left intact and that products that incorporated the code indicate in their documentation that the product contained code from the University of California and its contributors. Although Berkeley charged a \$1,000 fee to get a tape, anyone was free to get a copy from anyone who already had received it."

AT&T court case. The success of Networking Release 1 raised the question whether the whole operating system could be released in the same way. Another Berkeley computer scientist Keith Bostic started the project. He was able to attract volunteers to help him in rewriting the hundreds of AT&T copyrighted files. Programmers from different countries re-created the needed files by using the publicly available specifications.⁷⁰ After some more work to the kernel an almost full version of BSD carrying name "Network release 2." was released with a believe that it did not contain any AT&T's code and was placed under BSD license.

This was to be tested on the markets. Berkeley Software Design Inc. (BSDI) released this code as a commercial product titled BSD/386 adapted for the increasingly popular Intel 386 processor architecture in early 1992. It did not take long before Unix System Laboratories (USL), a majority-owned subsidiary of AT&T, sued them based on copyright and trade secret violations. Later USL also added University of California to the suit as a defendant. University of California promptly countersued USL since AT&T had also used Berkeley code in their Unix distribution and the parties locked up into a court fight.⁷¹

It soon became clear that both parties had made mistakes: AT&T's distributed files from BSD without proper copyright notices and BSD had still

⁷⁰ AT&T had published Unix specifications already in 1981

⁷¹ See USL v BSDI (1992) for details

a few files including some of AT&T's source code. The BSD dispute was solved after Novell bought USL in 1993. In January 1994, the settlement was finalized and the result was that three files were removed from BSD and another 70 were agreed to belong to USL.

Soon after the case was settled, BSD development split into different paths including FreeBSD, NetBSD and OpenBSD, who all run on the cheap PC architecture. It was now possible to freely develop and redistribute BSD Unix branches under the BSD license. However, the popularity contest for the preferred operating system of Internet servers was already lost. The legal case, open but closely coordinated development model, and now a split into different development paths guaranteed Linux a flying start as the new preferred Unix-based operating system for Internet servers.

2.3.2 GNU General Public License, Linux and SCO

Stallman invents GPL. Richard Stallman, a former staff member at MIT artificial intelligence lab, started his GNU project publishing the GNU manifesto in 1983 and then founding the Free Software Foundation. While GNU manifesto had a political and ideological tone, the project was at first technologically focused. The aim was no more or less than to write a complete Unix-compatible operating system. But it didn't take long for licenses to get into the picture.

When Stallman was starting the GNU project he worked on Emacs text editor. Stallman had written the first Emacs back in 1970s but because it was written in another programming language the source code was not useful. Subsequently James Goslig had written an Unix implementation of Emacs in 1982, which he distributed with source code. Stallman took Gosling's source code and started to modify it to become GNU Emacs. Meanwhile, Goslig sold his Emacs to a company, which claimed that Stallman wasn't allowed to distribute GNU Emacs because he had no authorization from the new copyright owner. In effect, Stallman was forced to rewrite all

code written by Gosling.⁷² Stallman memorized the event in a lecture in 1986:⁷³

"So it's sort of strange that they then changed their mind and refused to sign that agreement, and put up a message on the network saying that I wasn't allowed to distribute the program. They didn't actually say that they would do anything, they just said that it wasn't clear whether they might ever someday do something. And this was enough to scare people so that no one would use it any more, which is a sad thing."

This was just one of the many occasions when *Stallman was excluded* from the further development of an interesting project.⁷⁴ To end these "sad things" from happening, Stallman wrote Emacs General Public License in 1988. The idea of *copyleft* was for the first time implemented in this legal copyright license text, which held that GNU Emacs was not public domain but under copyright.⁷⁵ It was free to copy and distribute but it wasn't allowed to change the license terms in any derivative work. It is worth noting that before Emacs GPL Free Software Foundation didn't use any license for their software and Stallman appeared to be opposing copyrighting software.⁷⁶

With an innovative license Stallman was able to go against the exclusive effects of copyright with the help of copyright itself. In 1989 Emacs GPL license text was partly rewritten for clarity and the license was renamed to GNU General Public License. It became the default license for all GNU programs. The second version of GNU GPL was published in 1991 and the third version is currently in preparation.

⁷² See Tai (2001). Zawinski (2003) has gathered timeline describing Emacs development.

⁷³ Stallman (1986).

⁷⁴ Another anecdotal story is about buggy Xerox printer driver at MIT AI Lab. Stallman wanted to fix the driver but the developer refused to give source code because he worked under non-disclosure agreement (Williams 2001). One more is about LISP programming language, which was originally developed at the MIT AI Lab. Later on, MIT licensed the code exclusively to two companies who made it proprietary and excluded Stallman from open development.

⁷⁵ Later in this book we use the term "reciprocity obligation" when we speak of copyleft.

⁷⁶ Stallman (1986) for example said "I want to establish that the practice of owning software is both materially wasteful, spiritually harmful to society and evil" and "we are back in the same situation as in the ancient world where copyright did not make sense."

Linux and SCO court case. The breakthrough of GPL wasn't however Stallman's GNU software. It was Linus Torvald's new Unix-compatible operating system kernel, which he started to develop in 1991. In January 1992, Torvalds decided to license Linux with GPL.⁷⁷ The subsequent success of Linux accompanied with GNU and other free software meant that GPL license became more known and popular also outside hacker circles.

As with BSD, also Linux has faced legal charges of its origins by large Unix vendors. The next plaintiff was to be SCO, which bought Unix-business from Novell in 1995. Later SCO was acquired by Caldera, which again changed its name back to SCO in 2002. SCO/Caldera focused for some years to Open Source having an own Linux-distribution but they never really created any sustained business out of it.

In 2002 the company apparently decided that it made business sense to use their rights from the original AT&T Unix to initiate legal actions. SCO started to make statements about possible court cases against Linux supporters and finally sued IBM in March 2003 for one billion dollars, which was later raised to three billion. SCO claimed first that it owned the rights to all the features, which were added to systems such as Linux somehow relating to AT&T Unix System V. They claimed that IBM, having access to SCO's source code, had donated these features to Linux.⁷⁸

It eventually became evident that SCO's claims lacked factual basis. No source code in Linux was found to be copied from Unix System V. Novell disputed that SCO's ownership of Unix copyrights and patents would be clear. In addition, SCO as Caldera had distributed early Unix sources in 2002 under GPL license without restricting the development of derivative works in any sense. Thus, SCO had to back off from intellectual property claims and concentrate on possible contract violations. As of late 2004, the case is still going on.

⁷⁷ According to Torvalds, after the GPL decision he was "lying awake at night ... nervous about what commercial interests would do to the system." See Torvalds and Diamond (2001), p. 96

⁷⁸ See SCO v. IBM (2003)

2.3.3 Open Source Enters Vocabulary

Open Source Initiative. After the BSD case was settled and Linux became popular, it seemed that open source had finally passed the necessary “acid tests” for business credibility. The corporate interest in open source software was growing fast in par with the Internet boom. However, the BSD and GPL licensing models were still unknown to managers and even to most of the technical audience in the software industry. Corporations had also difficulties in understanding Stallman’s free software ideology.

Eric S. Raymond was the key individual who catalyzed the momentum into what became later called as the open source movement. His influential essay “The Cathedral and the Bazaar”, first published in 1997, and subsequent speeches attracted Netscape’s attention.⁷⁹ The web browser company was at the time losing market share to Microsoft and was eager to experiment with radical alternatives. Clearly separating himself from Stallman’s ideals, Raymond managed to persuade Netscape to adopt an open source strategy as the first well-known software company in January 1998.

Right after Netscape announced their move to open source, computer book publisher Tim O’Reilly organized a meeting for some of the most well-known open source developers to discuss a common public strategy.⁸⁰ Consequently, Open Source Initiative (OSI) was founded in February 1998 to address the increasing interest in Linux and other software developed under unifying open source principles. It started to certify licenses, which comply with the general terms of the Open Source Definition drafted by Bruce Perens.⁸¹

⁷⁹ In his essay, Raymond argues that Internet-based open source software development method as used for example in Linux (“Bazaar”) is fundamentally superior to an alternative, where development is restricted to a closed group (“Cathedral”). While originally Raymond’s criticism was targeted towards Stallman’s Cathedral-style free software projects such as Emacs, the terms were later generalized to contrast open source software (Bazaar) against proprietary software (Cathedral). The essay is reprinted for example in Raymond (2001).

⁸⁰ The list included e.g. Eric Allman (developer of Sendmail), Linus Torvalds (Linux), Paul Vixie (BIND), Guido van Rossum (Python) and Larry Wall (Perl). Notably, Stallman wasn’t listed. See Williams (2002), p. 162-163.

⁸¹ Open Source Definition was further edited from Debian Free Software Guidelines (now called as Debian Social Contract). For an overview of the events leading to the foundation of OSI see e.g. Williams (2002), p. 155-168, and Weber (2004), p. 111-115.

The software industry then took open source gradually into the big business. In June 1998 IBM announced it would support Apache and in July Oracle announced it will port its flagship database into Linux. In August Microsoft said officially that they are worried about Linux and Apache in particular. Trade press and popular newsmagazines started to cover open source success stories.⁸²

Industry reaction. Today, over six years later, we can say that open source has become a generally accepted practice to develop and distribute almost any kind of commercially viable software. The following excerpts from different company websites tell the level of enthusiasm in 2004:

IBM: "Linux is perfect for anyone with an eye on their budget who still needs a reliable and scalable operating system."
(ibm.com/linux)

Oracle: "With technical contributions to enhance Linux, with code-level support of the key Linux operating systems, and with strategic partnerships, Oracle is offering an Unbreakable Linux platform for customers to safely deploy Linux in a mission critical environment."
(otn.oracle.com/tech/linux)

HP: "HP is hosting a number of open source software projects that run on various HP systems." (opensource.hp.com)

Apple: "Apple's open source projects allow developers to customize and enhance key Apple software. Through the open source model, Apple engineers and the open source community collaborate to create better, faster and more reliable products for our users."
(developer.apple.com/darwin)

Microsoft. "The software industry often is depicted as ... rival camps of commercial and open-source providers. Market forces,

⁸² See e.g. OSI (1999) for an overview of major events during 1998-1999

however, are rendering this portrayal obsolete. Both models have proven beneficial to the software market.⁸³

Many *industry studies* have explored the potential practical benefits and risks of company migration to open source products. For example, Forrester Research interviewed in 2004 IT managers from fifty North American companies worth \$1 billion or more to name benefits and challenges of open source software:

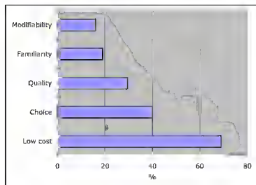


Figure 6. Benefits of open source according to IT manager interviews in 2004

⁸³ In 2001, when Microsoft publicly criticized open source heavily amidst the shared source initiative, their website read, "We believe that a shared source model, coupled with continuing contributions to public standards, provides a path that is preferable to the open source approach founded on the GPL."

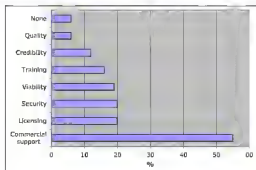


Figure 7. Challenges of open source according to IT manager interviews in 2004 ⁴⁴

Obviously, the low cost of licenses and avoidance of lock-in to one provider are major factors speaking for open source. Interestingly, also program quality is mentioned frequently; while an average quality of any published open source program may not be that high, the few star examples guarantee good public image for open source. Open source is also becoming more and more familiar since most computer science and engineering students nowadays study everything on open source. Also worth to note is that the practical possibility to modify the source code is not among the key criteria why open source is used.

Many challenges remain, though. Most frequently mentioned is the lack of commercial support. While major IT companies market open source as part of their hardware solution (such as IBM, HP or Apple), or as a platform for their proprietary software (such as Oracle and IBM) they may not always directly support particular open source products. Licensing comes second. Obviously licensing can be troublesome especially for those who develop or market their own software, which is based on open source. There are also many general challenges that supposedly are related to the

⁴⁴ LaMonica (2004), presenting data from Forrester Research. Naturally, one must judge the results based on the fact that the detailed interview data is not available from the commercial research corporation

fact open source products are still quite novel for many companies. These include security doubts, the overall viability and credibility of open source products and the lack of corporate training.

2.4 Social and Policy Dimensions of Open Source⁶⁵

Before going forward with theory, let's do a short side trip inside the developer communities and public policy debate. We aim to trace their ethical values, which form the basis of community norms and are further reflected in licenses.

2.4.1 Open Source and Individual Empowerment

In 1976 when software mass markets were still to form, Microsoft founder Bill Gates sent his now famous "An Open Letter to Hobbyists". In his letter Gates warned the early hackers ("hobbyists") not to freely distribute commercial software between each other. Gates did not believe hackers could ever have motivation to write any competing software to the products of software companies:

"Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software."⁶⁶

The letter was addressed to the members of the Homebrew Computing Club, an informal gathering of some of the first hackers in Silicon Valley in the mid 1970s.⁶⁷ Microsoft's first product ALTAIR Basic was distributed within the Club without Microsoft's authorization.⁶⁸

In a way, the world of Gates has turned upside-down during the last three decades. Software development tools and methodologies have stan-

⁶⁵ This part was originally published in Välimäki (2003b)

⁶⁶ Gates (1976)

⁶⁷ See Levy (1984), chapter 10, for a history of the Homebrew Computing Club

⁶⁸ Words: hacker and cracker should not be confused. In this study, we mean by hackers "computer programming enthusiasts". The word cracker is reserved for "illegal" hackers whose aim is to bypass security or copy protection measures with the means of hacking

standardized to an extent that distributed and freely organized development is finally efficient. Computer hobbyists have organized on the Internet and laid the foundations of a competing moral code.⁸⁹ Today one can speak of an open source and free software community working from garages and cellars into one pile. Large companies, now notably excluding Microsoft, support the work of the community and adopt its fruits to mainstream business models.

What has happened in between? One way is to look at it from the perspective of science. There have always been universities where the culture of free sharing and open development prevailed. Richard Stallman announced GNU project at the MIT in 1983 and open academic UNIX systems came from the hackers at the UC Berkeley. Also Linus Torvalds started Linux while studying computer science at the University of Helsinki:

"Back then, in '91, '90, most Internet people were at universities which meant that the whole philosophy on the Internet was fairly university-minded. The best reason for making Linux available at that time was probably just because it fit the culture."⁹⁰

In other words, free software is not only formally or accidentally connected with science but part of it.⁹¹ Universities were a natural place for open source methods to nurture. In a way, the structure of open source development method likens the logic of scientific discovery with continuous conjectures and refutations. Moreover, it includes elements from both human and calculative disciplines. This human, or artistic, element highlights the role of the individual participant's personal knowledge, commitment and passion.

Thus, the characterization of free software development as an ideally open, collaborative and cumulative process geared towards technical su-

⁸⁹ Levy (1984) formulated the ethic already in his seminal book on hackers. The postulates state for example that "all information should be free" and "mistrust authority - promote decentralization." Himanen (2001) discusses the philosophy and ethics behind this moral code.

⁹⁰ As quoted in Asami (2001)

⁹¹ See also Kelly (2001)

periority is necessary insufficient. Individuals count. This is common wisdom, both in the academic and business circles. Software development is only seldom arranged as a democratic process among peers – individuals may have dramatically different capabilities to develop software.⁹² Internet and open source have only made these rare few individuals stronger and more visible than ever.

If we believe in the power of the educated mind, we may conclude that the everyday practical ideals of tomorrow are set by the radical thinkers of today. Individuals from Richard M. Stallman to Eric S. Raymond and Linus Torvalds have all played a crucial part in launching an institutional change that no longer depends on particular individuals. Perhaps the crucial point was when key individuals in open source and free software communities understood the larger social and cultural context they were changing. With expert-level economic or technical arguments they would have never changed the accepted moral codes and other societal fundamentals of the information society.

Douglass C. North argues that new institutional arrangements will emerge when there is need for institutional change.⁹³ We consider there to be valid arguments that the international legal and moral codes on the ownership of the knowledge economy have reached such a point. There was a social need for institutional change, which open source licenses and the free software ideology are now fulfilling.⁹⁴

2.4.2 Community and Its Camps

The concepts of open source and freedom are ambivalent and they are, at the best, defined in the discourse.⁹⁵ Therefore we need to start from identifying the main participants, or opinion leaders, in the discussion.

In open source world, there is no fundamental difference between institutions (such as firms and states) and individuals. In principle anyone can

⁹² Brooks (1975) famously illustrated this in his seminal book on the project management lessons from IBM System/360.

⁹³ North (1982).

⁹⁴ For example Weber (2004), p. 179, argues that open source licenses are not only legal documents but also constitutional statements of certain social structures.

⁹⁵ See Berry (2004) for how discourse analysis can be applied to free / open source discussion.

participate into an open source project, even anonymously. No wonder, the number and variety of discussion participants is huge. In the words of Fink, "there are too many companies, too many groups and too many individuals for it [open source] to go away anytime soon".⁹⁶ However, amidst the noise, several voices are clearly louder than others.

Eric S. Raymond may be characterized as a sort of anthropologist of the open source community. Before writing "The Cathedral and the Bazaar" and starting to advocate the term open source, Raymond was already known in the hacker community for the maintenance of "Jargon File", a dictionary of the hacker slang. Since he co-founded Open Source Initiative in 1998, Raymond has been an active commentator and defendant of the corporate friendly image of the open source movement.

In heart of the community, there remains the figure of Richard M. Stallman. Of all the individuals in the community he is perhaps the closest one to an institution today. Power means also struggle. An observer who starts reading Stallman's writings and commentaries on the Internet soon finds out that the community is rather heterogeneous social system. There is the usual competition on leadership, mix of opinions, alliances, and such. But most of what is visible can be traced back to Stallman: Linux, GNU licenses and the concept of free software.

During the recent years Stallman has extended his concept of free software to free society.⁹⁷ This is mainly done through the Free Software Foundation. In the issues of public policy there are also other important participants with ideological debt and ties to free software and open source. Those include various grass roots movements and interests groups with Electronic Frontier Foundation in the lead.⁹⁸

In fact, one may argue that the most important cultural and political impacts coming from the open source and free software communities have nothing to do with the official open source movement. Open Source Initiative is not politically active but rather an informal interest group with practical and limited goals. However, while free software had been used

⁹⁶ Fink (2001)

⁹⁷ Stallman (2002)

⁹⁸ For more information on participants in the policy discussion see Beathwaite and Drahoš (2002) and Oksanen and Vähämäki (2002)

and developed for years OSI was the key effort that accelerated its breakthrough into the society at large. Without OSI, the political and cultural implications could have delayed even more.

2.4.3 Ethical or Technical Goals?

"I don't advocate open source" (Richard Stallman)

Let's now turn to the real discussion and compare the two ideological camps - free software and open source communities - with each other. What are they trying to say?

Stallman is not only interested in software and technical issues, but also policy, culture and ethics. In fact, the cultural and political aspects of free software seem to count him more. At heart, Stallman pushes GNU software licenses and his concept of freedom further than any technical issue. He clearly prioritizes, *a priori*, an ethical norm over a technical issue:

"Some free software is developed collaboratively, and some is not, but that is secondary... [Open Source] encourages people to think in terms of practical expediency, and it doesn't activate that part of their mind where they think about freedom and right and wrong and mistreating people or treating them right... They're putting the battle in the terrain where they think they can win on the question of just who is going to provide you with better software and support in the next few years..."⁹⁹

The open source camp may not reply. Linus Torvalds answers in short:

"I'm not religious about it. I think that Open Source is the almost certainly the best way to get the best technology."¹⁰⁰

⁹⁹ As quoted in Bowman (2002)

¹⁰⁰ As quoted in Asami (2001). On the other hand, Torvalds has recently expressed political motives sending open letters to SCO discussing Linux copyrights and another to European Parliament opposing software patents

Stallman ignores. In his rhetoric, Stallman is like writing to a community of voters who can either pick his or the competing - open source - ideology. Stallman compares the options to his audience:

"Let's compare the two philosophies. Open source gives priority to the developer's wishes; we give priority to what makes a community of free persons possible. The open source movement regards non-free software as a suboptimal solution; in the free software movement, non-free software is a problem - a social problem - and replacing it with free software is the solution. This is why corporations prefer "open source": because it doesn't raise issues they don't like."¹⁰¹

Stallman clearly indicates that proprietary software causes "a social problem" and it is the mission of free software community to fix it. In this sense, the open source community has more flexible approach. Trying not to polarize the issue too much, Eric Raymond takes a clearly opposing view to Stallman arguing:

"Explicitly jaw-boning people on issues like that I think is often counterproductive. The approach I'd prefer is to point it out fairly quietly that there are times when it makes economic sense to give up proprietary control, and lay out all the arguments and let people make their own conclusions."¹⁰²

2.4.4 Influencing Political Institutions

"It used to be that the problem was just the lack of free software, so we wrote free software. But now we face these attempts to pass laws that prohibit our work." (Richard Stallman)

The step from technical nuances into full blown political discourse has been taken. Open source has become increasingly political largely because

¹⁰¹ As quoted in Bowman (2002).

¹⁰² As quoted in Broesma (2002)

of the alliance of cyber rights activist groups with the academia and the spreading of Richard M. Stallman's ideology through GNU licensed software. It is not difficult to find a non-profit organization, mainstream politician or journalist who actively supports fair rights in information.

If one accepts Peter Drucker's characterization of non-profit organizations as "human change agents" who build up a community and define a common purpose, then Stallman and his Free Software Foundation are clearly winning ground.¹⁰³ The question now becomes how far they can go. Ultimately, Stallman believes free software can help in addressing fundamental societal problems.

"Free software can also help in solving other social problems. One of them is the digital divide, the fact that a large fraction of humanity can't afford to have access to any of this computer technology."¹⁰⁴

Interestingly, also Eric S. Raymond in the forefront of open source movement can be described as a sort of political fundamentalist. A non-compromisingly practical treatment of inherently political issues leads to what some call free market fundamentalism, others libertarianism.¹⁰⁵ Thus, one simply can't map the political opinions in the rhetoric of community opinion leaders within a particular political party or ism. There are libertarian elements of protecting individual freedom while, at the same time, one can find opposition towards the global corporate capitalism in side with socialist and green movements.

2.4.5 Practical Public Policy Initiatives

The political impact of open source is most visible in different public policy initiatives. A number of reports ranging from open source development policies to open source governmental policies have been pub-

¹⁰³ See Drucker (1990), p. xiv

¹⁰⁴ As quoted in Bowman (2002).

¹⁰⁵ As an extreme example, Raymond promotes an initiative called "gnus for geeks" claiming that people should have freedom not only to source code but to also own guns. "Geeks and guns are a natural match. Open source software is about getting freedom, personal firearms are about keeping it." See Raymond (2002)

lished during the last few years.¹⁰⁶ Some have hoped that open source would narrow the technological gap between rich and poor parts of the world. Others have strived for open source to be a preferred option in governmental software procurement policies. In addition, some argue that open source should be preferred for military because of vendor-independence. Already hundreds of open source related public policies and laws have been proposed and accepted in different parts of the world.¹⁰⁷

The purpose of this book is not to study any of these open source public policies. It must be noted, however, that historically the government has had a decisive role in the birth and growth of software industry in many countries. Governments are still large software buyers so public support for open source would also impact the software industry. No wonder, companies who have most to lose if open source becomes more popular have started to campaign explicitly against open source policies. Most notably, Microsoft launched in 2002 an Initiative for Software Choice, which argues among others that,¹⁰⁸

““Preference” laws harm the overwhelmingly proprietary-based IT industry because government administrators are instructed to automatically acquire or prefer OSS to proprietary offerings. More radically, “preference” proposals represent a fundamental assault on the incentive system that allows the IT industry to flourish and benefit consumers. Not only do they stymie competition in government markets, they signal to the industry that bedrock notions of free market enterprise, intellectual property protections, and the impetus to innovate no longer apply to their products and services.”

Open source advocates have reacted bitterly to such arguments.¹⁰⁹ Indeed, the above statement makes one ask whether open source really works against the “incentive system”, “competition”, “free market enter-

¹⁰⁶ See e.g. Hahn (2002) and Drahos (2003).

¹⁰⁷ See Keibet (2004) for a summary. Most policies support open source.

¹⁰⁸ Initiative for Software Choice (2004).

¹⁰⁹ See e.g. Perens (2002).

prise", "intellectual property" and innovation in the software industry. In the next chapter, we take a closer look at these economic issues behind the soft public policy argumentation.

2.5 Conclusion: Explaining the Increasing Role of Openness

Open source has been part of the software industry from the start. The idea of open source code was however hiding during the 1980s and early 1990s when the development and market conditions favored centralized and closed development and proprietary licensing. Still, shareware distribution model in mass markets and continuous open source development at universities showed that for some purposes it would be beneficial to distributed software freely and utilize an open development method.

In the 1990s, the environment eventually changed. We can now offer many explanations to open source's rise into the mainstream of the software industry. These include:

- From *technical* perspective, the rapid growth of the Internet, the trend towards cheap PC computers and a need for more flexible development methods have supported open source
- From *business* perspective, open source has offered possibilities for new companies to change the existing market structure and rules of the game. Open source has also altered the market strategies of incumbents.
- From *social policy* perspective, open source has been claimed to be a tool for more democracy, access to information and social equality as the role of software in the society continues to increase.

The social and cultural effects of open source can't be overemphasized. It is crucial for companies to understand the social and ethical dimensions if one wishes to really participate and influence open source development. Entering open source may also require fundamental reconsideration of company's intellectual property rights strategy.

We also noted that open source is a somewhat ambiguous concept itself. There is no singular open source community but rather multiple individual voices stemming from the hacker culture, whose leftist and liberal ideals originate from the 1970s. The software industry however adopted the term open source right after it was introduced and institutionalized in the late 1990s. In this book, we use open source much in the sense the software industry at large sees it as a reference to various open source software businesses.

3 ECONOMIC PRINCIPLES OF SOFTWARE PRODUCTS

This chapter discusses the economic theories of networks, copyright and patents in the context of software products. All theories characterize different aspects of software products bearing some differences but also many similarities. Aim of this chapter is to build a coherent theory where open source software licensing models can be analyzed within a larger economic context.

3.1 Economic Characterization of Software Products

3.1.1 A Network Economics Approach

In this study, we follow the industrial economists' approach analyzing software in the context of network industries.¹¹⁰ The viewpoint taken is of a firm producing software products to competitive markets. We are mainly interested in software as an economic product of the software industry. In the following, we explain how concepts developed in the literature of network economics apply to software product markets.

Unfortunately, this is not as straightforward as it sounds since it is difficult to judge which economic concepts are most relevant. The fact is, there is currently oversupply of fine-grained economic characterizations and classifications of software markets.¹¹¹ For example, Shy's textbook identifies four main attributes as compatibility and standards; network externalities; lock-in and switching costs; and economies of scale.¹¹² Gottinger lists sixteen "strategic characteristics".¹¹³ Moreover, there are sound and empirically backed arguments by e.g. Liebowitz and Margolis suggesting the general role and explanation power of the network economics approach to

¹¹⁰ Relevant theoretical research can be traced back to Katz and Shapiro (1985) discussing the nature of network effects and market power especially from the producer's perspective and Teece (1986) considering how also complementary producers may generate profits. A modern overview of the network economics approach as applied to the software industry is presented in e.g. Shapiro and Varian (1999), Shy (2001), and Messerschmitt and Szyperski (2003).

¹¹¹ Westrip (2003), p. 5.

¹¹² Shy (2001).

¹¹³ Gottinger (2003), pp. xv-xvi.

be limited in the context of software markets.¹¹⁴ While they acknowledge the theory to be relevant, the implications and usefulness of the results are not always valid in all market situations.

With these shortcomings in mind, let's go forward with the theory.

3.1.2 Software as an Economic Good

Software as an information good. The starting point for analysis is the economic theory of information. The assumption is that software can be described as a good, which is initially costly to produce but then cheap to reproduce. This means that the production of software implies what economists call the economics of scale. We can illustrate the production and reproduction costs of software products the following figure:

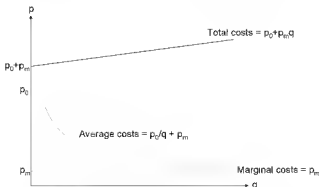


Figure 8. Different cost functions in the production of information goods¹¹⁵

In the figure p_0 denotes initial production costs and p_m costs from reproducing each additional copy. Total cost function is an aggregate of these two and average costs equals total costs divided by the number of copies made. The figure helps to explain that for any given price above p_m

¹¹⁴ Liebowitz and Margolis (2001), Liebowitz (2002).

¹¹⁵ Shy (2001), p. 54

there is a certain level of sales (q) after which every reproduced copy implies profit. Since the price of the product is not dependent on p_c , we can note that the development costs are not a good basis for software product pricing.

While the model is simple and explained in many economics textbooks, it has severe limitations of applicability in the software products industry. The model does not take into account the costs of selling and supporting the product; for example, the costs from marketing, maintenance, and upgrading are not included in “marginal costs”, which are assumed to be constant and minimal compared to production costs. Still, the model of software as information good may be applicable to e.g. entertainment software and other end-user applications with minimal selling and support effort from the side of the producer.

Software as a capital good. Capital goods are costly to produce and the product supplier earns profit from the installation, maintenance and support. Capital good theory has been applied to e.g. construction projects.

But also larger software products can be likened in many ways to capital goods. Especially early software products were very costly to produce and once the system was up, it was hoped to be around for years to come. Today for example Enterprise Resource Planning (ERP) software can be best described as capital goods.

One could think that the economic effects of copying depend on whether the work should be regarded as a consumer or a capital product. It is popular to claim, for instance, that the social costs of copying consumer goods are less damaging since the society consists mostly of individual consumers. It is also costly to monitor consumer behavior. Further, one can also argue that the diffusion of important technology in capital goods increases innovation and the social welfare as well. Especially when software products are used as means of production they should probably have more flexible use rights than assumed in copyright law. On the other hand, it is always possible to counterclaim that copying decreases the creation of new works whether they were consumer or capital goods.¹¹⁶

¹¹⁶ Watt (2000), p. 25–26.

Software as a public good. Independent of the theory of information and capital goods, software as a product can be also seen as a public good. For distinction between *public and private goods* two sub-concepts require definition:

- Non-excludable. There can be unlimited number of simultaneous users of the good; one can't restrict others from using the good. The quantity of the good is can't be controlled; it is equal to everyone. Examples of non-excludable goods are public spaces and open source software.¹¹⁷
- Non-rival. Goods do not consume; the usage of the good by one does not diminish its usability by others. The quality of the good is not controllable; it is equal to everyone. Typical examples of non-rival goods are air and proprietary software.

Public goods are both non-rival and non-excludable. Thus, their value does not diminish but instead increases from use¹¹⁸. Private goods, in comparison, are both rival and excludable. This can be perhaps best illustrated in the following table:

	Rival	Non-rival
Excludable	Private goods Private car	Proprietary software
Non-excludable	Public space	Public goods Open source software

Table 5 Public goods such as free software are non-excludable and non-rival

The theory of public goods sounds appealing in the context of open source software. Intellectual property rights, for their part, arguable privatize the public good nature of software by offering excludability. Still, also proprietary software is non-rival: copying software against its license

¹¹⁷ Non-excludable goods are often called *commons*. Selfish overuse of the commons results in the *tragedy of the commons*, a term coined by Hardin (1968).

¹¹⁸ Some authors call this feature of public goods as the *creed of the commons*. See e.g. Rose (1986).

terms (piracy) does not diminish its value to the user (though some complementary products or services such as warranty and support may not be available).

3.1.3 Components and Systems

Finally, software can be characterized as a system product. We know that software is used in computer systems consisting of different hardware and software components. Software is never used in isolation but as part of a system. The system may consist of separate hardware and software components and for the system to work the components must in some sense work together. This is, they must be compatible with each other. In this study, we define terms *compatibility*, *interoperability*, and *standards* as follows:

- A component is compatible with another if they communicate together without modification. Further, compatibility can be one-way or two-way, if both components can communicate with each other (two-way compatibility), then they are said to be interoperable. For example new software can be compatible with old software ("backwards" compatible) but not vice versa; in this case new and old software are not interoperable.
- A standard is an ex ante set of rules aimed at compatibility between components. Interoperable components follow the same standard. Further, *standards can be open or closed*. Anyone is in principle free to write an own implementation for an open standard.¹¹⁹ Obviously, an open standard must have open documentation and any test suites must be openly available. For example many programming and for-

¹¹⁹ Open standards should not be confused with the principles of open source in this regard. Open standards can traditionally include so-called reasonable and non-discriminatory (RAND) patent licensing terms, which are contrary to the principles of open source. Open source advocates demand that standards should be royalty-free to be called open while most industry standard groups do allow patent royalties. See e.g. Rosen (2004), pp. 304-311, W3C Patent Policy (2004) as an example of a recently adopted royalty-free standards definition and Kane (2002).

matting languages are clear examples of open standards. Closed standard is the complement of an open standard

Below is an illustration of the components approach to software:

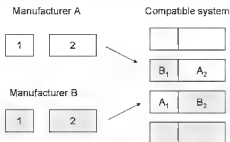


Figure 9 Components approach to software products

Manufacturers A and B both produce two components, here named as 1 and 2. If only combinations A_1A_2 and B_1B_2 are possible, then the components are incompatible. If either A_1B_2 or B_1A_2 is possible, then the components are one-way (or e.g. backwards) compatible. If all combinations are possible, then the components are two-way compatible and perfectly interoperable.

Compatibility can be gradual. A perfect compatibility would mean there are no costs or performance reduction between the communications between two components. In practice, for example Unix operating systems are more or less compatible with each other. While the Unix system core is standardized (open POSIX standard), each implementation follows the standard only partially or adds own incompatible features.

If components are owned by different companies, this may lead to *fragmentation*. If there are too many owners with exclusive rights for different components of the system, this may lead to underutilization because of the

costs for negotiating the necessary rights for the whole system may prove too high.¹²⁰

It is often thought that open source software must be compatible by its nature (available and modifiable source code) and follow open standards. However, that is not always the case. It is possible, as will be discussed later, that open source follows a closed standard.¹²¹ Further, some open source licenses may be incompatible with each other and thus the same fragmentation problems may arise as with any proprietary components.¹²²

3.1.4 Path Dependence, Lock-In and Network Effects

The system product approach can be used to explain why in the software industry only few products typically dominate the market at a time. In short, it may not be rational for users to change from one product path to another because of lost system compatibility or benefits of existing user network. This leads to define *switching costs and lock-in*:

- Switching costs are the costs of migration between incompatible components. We can further identify one-way or two-way switching costs. For example, if one needs to learn how new component works before using it, there will be no switching cost back to the old (already known) component. To contrast, any cost in the actual switching transaction means that there will be also costs in switching back.¹²³
- Recently, switching costs have been analyzed as a tool to create lock-in situations. If the switching costs are high enough to affect the purchase choice of the user in favor of the product he is currently using, then we say there is a lock-in situation.¹²⁴ Lock-in can be for example contractual, related to durable purchases or based on loyalty programs.¹²⁵ Both hardware and software lock-in have been common. In

¹²⁰ This has been referred to as the *problem of anti-commons*, a term coined by Heller (1996)

¹²¹ Section 5.5.2

¹²² Section 5.3.4

¹²³ Farrell and Klemperer (2001)

¹²⁴ See e.g. Besen and Farrell (1994), Arthur (1989) and Katz and Shapiro (1985).

¹²⁵ Shapiro and Varian (1999), p. 117

principle, the effects of hardware lock-in can be fought with open systems architecture and software lock-in with open source software.

A common argument for open source migration is that it does not imply lock-in to particular developers. In theory, it is indeed possible to change the developers if source code is available and freely modifiable. However, the development of particular source code further may usually require the tacit knowledge of the original developers, which may be very costly to acquire especially if the product is very complex.¹²⁶ Moreover, as will be noted later, the license terms may still include limitations, which later result in a *de facto* lock-in situation.

Finally, positive *network effects* also characterize software markets:¹²⁷

- If the value of the good to its user depends on the number of other users, the good is said to have network effects.¹²⁸ In software markets, network effects are typically positive: large user base raises the value of the product to one individual.
- Network effects can be both direct and indirect. Effects to the product itself are said to be direct and effects to other products through compatibility or non-compatibility are said to be indirect.¹²⁹
- Network effects are not restricted to material goods. It has been argued that for example not only software but also capabilities of the developers are shared and enjoy benefits of the expanding network.¹³⁰

In practice it may not be always appropriate to speak of *network externalities*, since software producers typically have the option of internalizing the outcome of network effects through e.g. proprietary copyright or patent

¹²⁶ For example the Mozilla and OpenOffice projects faced this kind of problems after they were open sourced. It took a long time to get new developers for such large and complex projects.

¹²⁷ Von Westarp (2003), p. 100-103, lists six empirical studies where the existence of network effects has been studied and proved in software, hardware and telecommunications industry. He also identifies numerous theoretical models for the detailed analysis of network effects.

¹²⁸ Another term is "bandwagon effects", a term proposed by Weblen and first analyzed by Leibenstein (1950).

¹²⁹ See e.g. Katz and Shapiro (1985), Economides (1996).

¹³⁰ See Garzavelli (2003), referring to the economic theory of professional clubs

licensing. However, in the case of open source, the software developers have in many cases no power to internalize the effects through such direct means. Thus, one can argue that for example the copying and use of GNU/Linux operating system kernel – whose copyright is permanently under an open source license – causes network externalities to companies competing in operating system software markets.¹³¹

3.2 Economics of Software Copyright

Does the society really need a copyright law in the first place? This age old question is as relevant today as it was when the printing press was new technology and the first privileges were granted in medieval European city-states. Rampant copying of music and movies on the Internet has made entertainment industry to call for stricter copyright enforcement while critics argue copyright should be reformed for digital works. In the midst of this debate, however, software copyright seems to work surprisingly well.

3.2.1 Motivation of Developers

While law treats copyright as a form of artistic creation, economist typically speaks of innovation. And no creative innovation will ever impact human society unless entrepreneurs and traders start distributing it in the society. However, the basic question of why creativity happens in the first place is still unanswered. For example the following reasons have been offered:¹³²

- Monetary and other material reward
- Fun and “scratch of the itch”¹³³
- Fame and merit
- Service to the society or mankind¹³⁴

¹³¹ See e.g. the opening example of Rahnasto (2003), p. 1

¹³² For an early list of reasons by an economist see Machlup (1962), p. 144–145.

¹³³ See e.g. Torvalds and Diamond (2001) and Raymond (2001).

¹³⁴ See e.g. Hamanen (2001) for extensive discussion on the “hacker ethic”

- Instinct of workmanship

Also one of the most discussed topics in open source has been the motivation of developers. Why do some developers contribute their spare time to community projects such as Linux or Apache? Answers ranging from the economics of signaling to ethical goals have been offered.¹³⁵ For the purposes of this book, it is sufficient to note that the software industry supports open source strongly. Commercial open source projects and other corporate interest in open source development shows that there are also clear industrial profit motives in addition to the possibly altruistic motives of individuals

3.2.2 Investors and Incentives

There are no definite answers how to increase innovative and creative activities in the society. For instance Machlup has noted that that the increase in inventor compensation, the number of inventors, or the number of innovations do not correspond directly to the number of new inventors and effective innovations.¹³⁶ It is especially not clear how creativity is linked with ownership and intellectual property.

The history of copyright shows two motives behind an institution for limited ownership over creative works. Until modern times, an important goal of copyright was to give both printing press owners and the state some control on what was published and printed. Thus, instead of creating incentives to new authors or innovators, the first privileges predating modern copyright laws started from the idea that the interests of those who *invested in the copying machines* and the supportive institutional structures should be protected in the first place.

The role of the author didn't rise until another few hundred years. John Locke is usually referred to as the first philosopher, who explained the dual existence of both commons and individual intellectual property.¹³⁷

¹³⁵ See e.g. Lerner and Tirole (2002) for signaling and Raymond (2001) for more altruistic or technical motivations

¹³⁶ Machlup (1962), p. 166

¹³⁷ Locke (1690), part V.

The idea was that every individual member of the social community should have the right to his own person. Further, all work and its results can be traced back to the person. Initially, all property belongs to commons but when an individual mixes his work with the commons this becomes his personal property.

Theories of property rights are tied to the cultural background. For example in many Asian cultures the copying of intellectual creations has been acceptable from early times. In China, the value of an artist may have been directly dependent on how much his works were imitated and copied.¹³⁸

In the 1700s copyright laws subsequently recognized the interests of individual authors and the institution was justified as *an incentive for authors* to create more new works.¹³⁹ However, copyright also continued to develop with close contact to technological development and the interests of investors. During the 20th century, copyright came to include maps, technical drawings, computer programs, and more recently also technical protection mechanisms – all products of industrial enterprises rather than individual authors. Roughly put, copyright followed technological change and at some point of time the early distinction between authors and investors became blurred.

In effect, one can argue that copyright is today justified both as an incentive to individual authors and as an institutional mean to protect creative investments. One obvious reason for the relative strength of the investment justification is the strengthening of the property analogy. Today, copyright and the management of intellectual property is regarded as a relevant part of any technology and media company strategy.¹⁴⁰

¹³⁸ See Alford (1995) and Stille (2002).

¹³⁹ The incentive theory is perhaps most famously manifested in the US Constitution, which stated an exclusive right to authors and inventors “for limited times” in order to “promote the progress of science and useful arts”.

¹⁴⁰ See e.g. Granstrand (1999) and Rahmasto (2003).

3.2.3 Costs of Copying

But how much does copying in fact cost to authors, investors and the society at large? Maybe the answer is: not that much. Already Plant argued famously that copyright law is not an optimal way to regulate the copying of books.¹⁴¹ Instead, he tried to show that even in the absence of copyright new books would be created, book publishing industry would exist and the society overall would be better off. He didn't however take into account the welfare losses to individual authors and publishers.

Term "piracy" has been used for centuries to characterize illegal copying.¹⁴² A recent IDC study commissioned by Business Software Alliance (BSA) calculated that 36% of software in use worldwide in 2003 was pirate copies. In the main markets this so-called piracy rate has been going down slowly: in the US the rate was 22% and Western Europe 26%. From this data, the study also calculates "losses" to software vendors by assuming that each illegal copy would have been bought legally at the market price.¹⁴³

Most economists have disagreed on such loss calculation for as long as such piracy studies have been published.¹⁴⁴ The economic problem with thinking copying as being categorically detrimental is roughly the following: the one who copies does not take anything exclusive away from the copyright holder but merely imitates and reproduces. The abstract work as an intellectual object does not change hands; only its real-world representations multiply. Now whether this multiplication increases or decreases the welfare of the authors, users and the society as a whole is a completely separate issue.

Thus it is not possible to prove in any way whether illegal copying causes such costs, which are commonly claimed by organizations like BSA.¹⁴⁵ Instead, economists have concentrated on building models to esti-

¹⁴¹ Plant (1934)

¹⁴² Already e.g. Luckombe (1771), p. 76, mentions piracy referring to the detrimental quality of copies.

¹⁴³ Business Software Alliance (2004).

¹⁴⁴ BSA was founded in 1987.

¹⁴⁵ This is one of the main results of Watt's comprehensive survey of the economic literature on copyright. See Watt (2000)

mate the optimal (illegal) copying rate from author's perspective. The assumption is that neither extreme (0 or 100% piracy rate) is beneficial to the authors. Allowing copying to some extent builds market base for the work in side with increasing the welfare of the society as a whole.¹⁴⁶ In a highly competitive market setting new entrants should obviously tolerate more piracy whereas incumbents should fight against it.¹⁴⁷

3.2.4 Optimal Limits of Copyright

It seems clear that copyright is not a typical property right. In addition to protect the interests of the author and the investor, copyright must also *allow users to access and modify works* in a reasonably liberal way. It is said that no new works are created without learning from others. In short, copyright must balance the benefits from the creation of new works with the costs of limiting free usage of those works.¹⁴⁸

Riis separates three components in copyright, which mainly determine its scope: rights covered, the scope of those rights and the term of copyright.¹⁴⁹ Let's discuss each of these in turn starting from the question whether it is beneficial for an author to allow indirect copying or not.

Indirect Copying. Economists have made difference between direct and indirect copying (imitation). When a work is modified or mixed with another work, it is difficult to find out what is the economic impact of this new work to the copyright holder of the original.¹⁵⁰ Perhaps indirect copying should be therefore regulated more flexibly taking into account the case-by-case economic effects to the original work's copyright holder.

The value of indirect copying is naturally different to the user and original author. For example, Besen and Raskind argue that that the original author should have only very limited right to influence modifications or derivative works. Giving imitators possibilities to create new works, which

¹⁴⁶ See Corner and Rumelt (1991) focusing on private returns and Takeyama (1994b) expanding the analysis to social welfare.

¹⁴⁷ See e.g. Prasada and Mahajan (2003) for an analysis.

¹⁴⁸ It can be assumed that the lost sales to copyright owners are fully compensated with increased free usage.

¹⁴⁹ Riis (1996), pp. 86-87.

¹⁵⁰ Watt (2000), p. 31-33.

are based on existing ones, one supports entrepreneurship and avoids costly reinvention.¹⁵¹ Also Landes and Posner point out that whenever new works are created, old ones are commonly used as the basis – explicitly and implicitly, by adding new contribution or literally. Obviously broader copyright increases the costs of creating new works if the existing authors are given power to control the creation of indirect copies or modified works.¹⁵²

Here, one can also point out Liebowitz's idea of applying the concept of price discrimination in copyright. In short, the copyright holder can license the work with tailored price and rights to different usage groups according to their need to e.g. make direct or indirect copies. Part of the users may be willing to pay more for the possibility to copy or modify the work further while it may not be reasonable to charge the majority but a minimal licensing fee if any.¹⁵³ Especially in the case the market implies network effects, copyright holder should consider first expanding market base by allowing copying and then later price discriminate those users who are most willing accept copyright enforcement and pay license fees for some types of uses.¹⁵⁴

Exceptions to Copyright. From economic perspective, any exemption in the scope of a right is justified when the costs of enforcing the right outweigh possible benefit. For example, private use of a work is typically allowed as a special exemption. The private use of a work would be very costly to enforce and, on the other hand, private use may directly affect the output of individual authors when they study existing works in creative ways.

In the context of computer programs, reverse engineering and interfaces have been debated issues. Should the scope of copyright reach also pro-

¹⁵¹ Besen and Raskind (1991).

¹⁵² Landes and Posner (1989), p. 333. On the other hand (p. 332), Landes and Posner also point out that limiting the author's control according to copyright law may lead to increasing self-protection through e.g. technical copy protection systems and marketing of inferior, quickly-prepared products in the hope that demand would be satisfied before imitators enter the market. One can ask, though, aren't publishers going into this direction without the scope of copyright having anything to do with the issue (since copyright is being broadened all the time).

¹⁵³ Liebowitz (1985).

¹⁵⁴ See Takeyama (1994b) for a model and discussion on long-term strategic implications.

gram interfaces? If not, should reverse engineering for interoperability, which may require the copying and modification of the program, be allowed as a special exemption to copyright? There is no one right answer. Reverse engineering the program may boost the creation of new works but on the other hand it may result in unfairly competing works and less incentives to create in the first hand.

Copyright's Term. In their influential study report to the US Copyright commission before computer programs were copyrighted, Braunstein et al noted that the optimal protection period should be judged case-by-case. As a rule of thumb, the protection time should be less than the economic life of the work. Now regarding copyright law, a crucial question is whether efficiency gains would outweigh administrative costs of multiple protection terms in copyright legislation.¹⁵⁵

Tailored copyright term is rational if we assume that the goal of copyright is a balanced benefit to both copyright owners and users; before copyright expires, owners have a temporary monopoly. In the case of software, product life-cycles have historically been somewhere between 5 and 15 years.¹⁵⁶ Still, copyright protection of computer programs follows today the same terms as with any artistic work and is currently multiple times over product lifetimes.

From an economic perspective, increasing the copyright term adds the transaction costs for copying and otherwise reusing old works, which are no more circulated but still under copyright. It may be difficult to contact the author or publisher of a work published over fifty years ago and negotiate a permission to copy the work. Instead, there is hardly any increase in the economic incentive for authors to create new works whether the copyright lasts 50 or 70 years after the death of the author.¹⁵⁷

Luckily, theoretically inefficiently long term of copyright has not slowed down new software development in practice. Since product life-cycles are

¹⁵⁵ Braunstein (1978), pp. 241-242, especially footnote 11. Already Plaut (1934) argued that since publishers take books typically out of circulation in 3-5 years, a copyright term of 25 years (as it was in the 1930s) is obviously economically inefficient.

¹⁵⁶ See Campbell-Kelly (2003) for a historical overview.

¹⁵⁷ Recently leading US economists published a well-argued note in a legal case challenging the copyright term extension in the United States from 50 to 70 years. See e.g. Akerlof et al (2002).

short and new technology breakthroughs common, essential parts of new products are time-by-time developed from scratch. Therefore any legacy technology and its copyright or other legal protection may not affect the development of new products in any way. One may speculate that perhaps implementing variable copyright terms to software works would not have implied such efficiency gains some economists have suggested.

The life-cycle model doesn't apply to computer games. Even the very first ever developed games can be played, and in fact also are played, on new computers through emulators.¹⁵⁸ It is in strict legal sense illegal to download most retro-games since copyrights to these games have not expired and many times the publishing company doesn't exist either with whom to negotiate. Retro-game hobbyists call such games abandonware and argue for releasing them from copyright. For example open sourcing abandonware is not possible if the copyright situation is unclear and so the only option to reduce the legal risks of distributing and copying such games would be to amend the copyright's term.

3.2.5 Compensation Mechanisms

There are numerous ways to receive compensation from copyrighted works. Maybe the most evident is to collect direct fees for all restricted acts as defined in the copyright law. A good example here is the model presented by Landes as Posner where copyright holder collects licensing fees (royalties) from restricted acts such as copying and distribution.¹⁵⁹

Licensing fees can be also addressed through a government-controlled system. The idea is to collect some kind of taxes from all users and then divide them according to some democratically decided criteria. This is for example the model used by copyright collecting societies for composers and recording artists.

¹⁵⁸ One of the most popular classic game website on the Internet, Home of the Underdogs, distributes currently over 4000 more than five year old games, which are still playable today. See <http://www.the-underdogs.org/>

¹⁵⁹ Landes and Posner (1989)

In large scale collective compensation systems were used in the former socialist countries where they covered basically all kind of works. Although it is difficult to judge the relevance of socialist copyright to the recent discussion on digital copyright, it is interesting to learn how their system was designed. In Soviet Union authors were compensated according to royalties set in the government regulation. Factors that affected the royalty rate of e.g. books were its length, genre, copies printed, and the quality determined by the state. Royalty rate didn't depend on the price or copies sold.¹⁶⁰

The soviet system aimed to compensate authors as other workers based mainly on the quality and quantity of their output. Unfortunately, the system didn't work in practice that way. Secondary issues like the number of characters in the work became sometimes determining factors. There was also a clear incentive to overprinting.¹⁶¹ Nor was the system equal; in practice some well known and government favored authors were exempted from the state set schedule and received higher royalties compared to others.¹⁶² So while the pricing system based on private royalties might not work optimally, it is difficult to find any historical evidence supporting collective compensation systems either.

Compensation systems based solely on licensing fees are, however, easy to criticize. In fact authors receive a substantial part of their compensation through other more indirect mechanisms than licensing fees. One way to categorize these alternative indirect compensation mechanisms is:¹⁶³

1. *Making copying expensive* The idea is that the one who possesses an intellectual creation has the possibility to technically protect access to his creation through different "barbed-wire" constructions. Even if the copying of the work is illegal, the author may

¹⁶⁰ On the Soviet remuneration system see e.g. Newcity (1978), p. 84.

¹⁶¹ In the 1970s Soviet Union printed slightly more books than the US. Ibid., p. v (referring to United Nations' statistics)

¹⁶² Ibid. p. 92

¹⁶³ See e.g. Novos and Waldman (1986), p. 10-16, Palmer (1989) and Watt (2000), p. 54-58

have an incentive to make copying more difficult because of the limited scope of copyright and obvious enforcement problems.¹⁶⁴

2. *Complementary goods.* Another product may be a necessary condition for the use of the work or an optional profit-generating feature or service; thus their sales are bundled. Also advertisements can be counted in this category.
3. *Timing.* Author may control the timing of the work introduction and make advance contracts before anything is created. Also, the author may try to introduce new versions of the work so quickly that copies are always out-of-date.
4. *Pricing and quality control.* If illegal copies cost more and are substantially of inferior quality than legal copies, then there is little incentive to copy. Pricing low can be an efficient strategy especially when the market has positive network effects.

Open source businesses were analyzed first by Raymond as alternatives to the licensing fee model. He argued that there existed open source businesses based on complementary services and support, complementary goods such as manual and hardware, trademark licensing and gaining market share for proprietary products.¹⁶⁵ These essentially cover complementary goods, timing and pricing from the indirect compensation alternatives listed above.

3.2.6 Is Software Copyright Inefficient?

In the recent years, a number of reform proposals for copyright have been suggested. Some of the most prominent include

1. *Patent-like copyright.* Essentially, copyright would require registration and would have a shorter term. In a way, this would make copyright more like real property, but for a strictly limited term.¹⁶⁶

¹⁶⁴ See Besen (1987) for the first comprehensive economic analysis of technical protection mechanisms.

¹⁶⁵ Raymond (2001).

¹⁶⁶ See e.g. Lessig (2001)

- 2 *Alternative compensation mechanisms.* As discussed in the previous section, these include in addition to the private reward systems also collective government-enforced systems such as hardware levies – which are currently widely used in Europe for other works but software – and compulsory licenses.¹⁶⁷
- 3 *Abolition of copyright.* Some technology activists keep on arguing for the cause.¹⁶⁸ In addition, some academics skeptical of the efficiency of copyright use such arguments if not directly then in between the lines.¹⁶⁹

It can be argued that these reform approaches do not aim for a more balanced copyright but merely to some extreme. Maybe the copyright discussion lives on just because there are extremist proposals from both those who advocate for more protection and those who advocate for copyright reform.

It must be stressed that the core of the copyright reform discussion concerns mainly the *entertainment industries*. From the perspective of software industry, these proposals have less validity. As noted, the term of software copyright is not relevant if the life-cycle of programs is short. Private compensation mechanisms from licensing fees to alternative models for software in fact work for software products as compared to music and movies. Piracy rates at around 30% tell that despite the fact that almost any program can be downloaded for free from peer-to-peer networks software users simply don't do that at such a scale that would render licensing busi-

¹⁶⁷ For these collective means, see e.g. Fisher (2004). Again, just like the soviet copyright system, hardware levies are based on the claim that it becomes cheaper for the society to organize a collective and centralized compensation mechanism rather than monitor and enforce private license agreements between individual economic agents. Specific problems with levies are that they do not affect directly the behavior of economic agents (copying of works) and, in addition, levies may have unwanted consequences (less recording capacity in use). In short, it can be questioned why users should pay in the first place to authors in order for them to avoid their possible future costs from illegal copying.

¹⁶⁸ For a modern classic rant, see Barlow (1994).

¹⁶⁹ E.g. Lemley (2004).

ness obsolete.¹⁷⁰ On the contrary, the absence of copyright collecting societies in the software industry suggests that even individual authors and small software companies can effectively license their copyrighted software products to mass markets.

3.3 Economics of Software Innovation and Patents

Economists have often compared software production to industrial innovation. A traditional argument support patents as *ex ante* incentive to appropriate innovation *ex post*. A closer look at software innovation however shows that patents are used rather as strategic assets to influence competitors than means to appropriate innovation. Optimal software innovation may in fact follow an open and collaborative innovation paradigm where innovation is appropriated through alternative means.

3.3.1 Innovation in the Software Industry

It has been questioned whether there is innovative activity in software development in the first place. Many well-known programmers do not regard themselves as innovators but rather as authors. For example, the creators of major modern operating systems have compared the design and development of operating system software to writing.¹⁷¹ They don't think that new software innovation is "discovered" but merely implemented.

For the purposes of this study, let's however assume that software development can be analyzed in the terms of innovative activity as most economists do. For example Torrisi identifies in his empirical study three categories of software companies from innovation management perspective:¹⁷²

¹⁷⁰ Of course, in traditional sale of goods a loss rate of 1/3 would be devastating. In an information good licensing business, however, the rate is viable. As was noted, the copyright holder may in fact benefit from increased user base if he develops alternative means to benefit from network effects.

¹⁷¹ See for example Conner (1998) for Tim Patterson's comments (MS DOS) and Stapleton (2004) for Richard Stallman's comments (GNU/Linux). It must be noted that some programmers may emphasize the liberal aspect of software development simply because it is a way to object software patents. Patents can be granted only to "innovations" as defined in patent laws.

¹⁷² Torrisi (1998), pp. 122-126 and pp. 162-163.

1. Entrepreneurial start-ups (about 5 years old) These companies *create new innovations*. They are operated by flexible organizations and do not use engineering techniques since it is difficult to codify their practices.
2. Small and medium sized companies specializing in few products or services (more than 5 years old with less than 500 employees). They make *incremental product innovation* (quality, differentiation) and mainly contribute in the *diffusion of innovations*. These firms rely on centralized management.¹⁷³
3. Large corporations offering system software and services (more than 500 employees). Their role is in the *creative blending of innovations* from various sources. Corporations rely on formal rules in innovation management.

Innovation in the software industry can be characterized as *serial and cumulative*:

- Serial (or continuous) innovation builds on the existing innovations.
- Cumulative (or incremental, sequential) innovation is based on separate small innovations that together build the product together. In computer technology, many innovations come from complementary technologies.¹⁷⁴ Cumulative innovation implies *fragmentation* problem.

We can also distinguish between *continuous and radical* innovation:

- Continuous innovation builds on the top of existing innovation making better technology in small steps

¹⁷³ According to Torrisi, most European firms belong to this category. He also notes in pp 158-159 that between 1990 and 1997 European firms have shifted their focus to new product innovation and both copyright and lead time have increased importance as tools to appropriate rent from innovations.

¹⁷⁴ Torrisi (1998), p. 112

- Radical (or disruptive or revolutionary) innovation completely changes the market structure¹⁷⁵

Technically, software products typically evolve through number of minimal new innovations, which are built and depend on previous knowledge. Radically new innovative products that change the markets fundamentally appear more likely from long socio-economic processes. It is also relevant to understand the context or level where such radical innovations come from. In a networked environment, the role of users as innovators becomes central and the overall social aspects of innovative activity are strengthened.¹⁷⁶ Thus, networked innovation also increases interdependencies and cross-licensing possibilities between different types of companies.¹⁷⁷

3.3.2 Difficult Relationship Between Innovation and Patents

Schumpeter famously explained innovation with legal and other institutional structures that create incentives.¹⁷⁸ Economist then went on to argue that strong patents are the institution, which optimally maximizes innovation.¹⁷⁹ The innovation argument was difficult to criticize. Without innovative activity and entrepreneurship the technical progress in society would stall and no new value would be created. Hardly anyone wants to stop innovative activity. It seems, however, that academics now disagree on the criteria for institutional structures that really promote innovative activity. Strong patents in particular have been questioned.

Already in 1958, Machlup pointed out famously in his seminal study on the US patent system as follows:

¹⁷⁵ Christenson (1997) Already Dosi (1982), pp 151-157, presented a theory of technological development through disruptive innovations (using term "radical technological change") following Kuhn's (1968) classic presentation on scientific revolutions

¹⁷⁶ See von Hippel (1988) generally on users as innovators and von Hippel (2002) specifically on open source users as innovators Tuomi (2002) has discussed extensively the social aspects of innovation networks

¹⁷⁷ Teece (1986)

¹⁷⁸ Schumpeter (1942)

¹⁷⁹ E.g. Nordhaus (1969) argued for strong patents because of their ex ante incentives. Kitch's (1977) prospect theory further suggested that strong patents have several ex post benefits such as efficient coordination of future innovations and reduction of transaction costs

"None of the empirical evidence at our disposal and none of the theoretical arguments presented either confirms or confutes the belief that the patent system has promoted the progress of the technical arts and the productivity of the economy."

Today, debate on the economic effects of patents is furious. Granted, historically computer and software industries have had rather weak patent protection and still nobody has questioned that the fields have been very innovative. Some economists have presented empirical evidence against strong patents especially in the software industry. It seems that in those fields of technology, where patents have been taken into use, investments to research and development have slowed down.¹⁸⁰ Other economists defend software patents arguing that the major issues are in the patent quality. According to them, patents are a major incentive for innovation also in the software industry.¹⁸¹

Interestingly, open source software provides one more argument refuting the necessary connection between innovation and patents. Also open source licenses can be seen as institutional structures for creating new information and innovations. In the open source world, the most significant resource of innovations is individual participants, and the use of patents is explicitly disallowed.¹⁸² The numerous possible means of individuals to share and exploit their tacit knowledge affects undoubtedly innovative processes. Thus, one could argue for a Hayekian explanation for an open source innovation where individuals spontaneously cooperate within a structure governed by the license in question.¹⁸³

To conclude, whether patents promote innovation or not, they have certainly other functions as well. For example Levin et al noted from extensive interviews that many companies patent to monitor who are the innovators within the company.¹⁸⁴ Also, patents may be collected for purely regulatory reasons; for instance some developing countries have required

¹⁸⁰ E.g. Bessen and Maskin (2004)

¹⁸¹ E.g. Jaffe and Lerner (2004)

¹⁸² von Hippel (2002)

¹⁸³ Hayek (1945).

¹⁸⁴ Levin et al (1987)

technology licensing in order to enter their markets.¹⁸⁵ But most often, patents are often analyzed in the context of corporate strategy where their function is to leverage market power by affecting the behavior of other companies.

3.3.3 Patents as Strategic Assets

In recent years, numerous studies have been published on the actual use of patents in the software industry.¹⁸⁶ Large IT companies currently apply for more patents than their counterparts in other technical fields. However, software patents are used not only to create licensing income but they are increasingly used for *strategic reasons*. Collecting strategic patent portfolios can be use in e.g. signaling to investors, entry barriers to new entrants, offensive or defensive assets in litigation and for cross-licensing purposes.¹⁸⁷

Stac v. Microsoft is an example of strategic use of software patents. Microsoft settled with Stac Electronics over a software patent issue in 1994. Stac had filed a suit alleging Microsoft infringed their disk storage patents.¹⁸⁸ After jury had awarded Stac 120 million dollars in damages, the case was settled with Microsoft paying Stac 40 million and buying their stock. The case has been viewed both as an example of a successful software patent protecting a new innovation and as a strategic legal effort of a struggling company to stay alive at the cost of technological development. Stac Electronics went out of business soon afterwards mainly because they lost their technological leadership and it was rather easy to “invent around” their patent portfolio

The strategic nature of patents is strengthened by the fact that in order for patents to be useful in practice, companies need to have plenty of them.

¹⁸⁵ This is an interesting result as it is generally known that the costs of absorbing innovation can be high; obviously the patent system does not help in this regard as it should.

¹⁸⁶ It must be noted that many of the recent studies are connected to the policy discussion of if and to what extent innovations implemented in software should be patentable. Because of this close policy connection, some studies may be seriously flawed.

¹⁸⁷ Besen and Hunt (2004)

¹⁸⁸ See *Stac v. Microsoft* (1993).

It has been estimated, that in the United States around 15% of all patents are ever litigated and as few as 0.1% are litigated until trial.¹⁸⁹ Further, as many as 46% of all patents litigated to trial are finally held invalid.¹⁹⁰ Thus, Lemley and Shapiro have characterized patents not as exclusive but rather as “probabilistic” property rights.¹⁹¹ Since not every patent is truly valuable, companies have an incentive to build as large patent portfolios as possible if they ever want to enter the patenting game.

Finally, we must note that increasing patenting by big companies does not necessarily mean that small software companies or independent developers would suffer and innovation would be stalled. As we will see later, it seems that those who have patents mainly use them against each other if the “patenting warfare” ever enters an active state. The dynamics of the patenting game includes the rather surprising finding that big companies protect for example open source developers with their patents.¹⁹²

3.3.4 Different Means to Appropriate Innovation

Obviously, the appropriation of software is complex since software is cheap to replicate and the intellectual property protection and enforcement is far from complete.¹⁹³ According to Torrisi’s empirical study, legal tools are specifically considered a weak instrument of appropriability in the software industry.¹⁹⁴ In addition to legal means such as copyright and patents, at least the following methods are possible for appropriating returns from software innovations:¹⁹⁵

1. *Lead time.* Being first-to-market is according to empirical studies the most important way to appropriate returns. Strong sales and market-

¹⁸⁹ Lanjouw and Schankermann (2001) and Lemley (2001).

¹⁹⁰ Allison and Lemley (1998).

¹⁹¹ See Lemley and Shapiro (2004) pointing out that many economic theories on patents build on the unrealistic assumption that patents would be well-defined exclusive property rights.

¹⁹² Chapter 6.

¹⁹³ See Teece (2000), pp. 16–19, who differs between weak – moderate – and strong appropriability. He argues that hard replicability and strong intellectual property protection increase appropriability conditions.

¹⁹⁴ Torrisi (1998), p. 109/110. However, the importance of legal protection was growing during the 1990s and it was found more important than secrecy.

¹⁹⁵ Levin et al (1987), Torrisi (1998).

ing efforts may be essential to establish a strong position over followers

2. *Continuous improvement.* Close to the lead time alternative, being technically (in quality) ahead the competition also impacts returns according to empirical studies.¹⁹⁶
3. *Skilled personnel.* It can be argued that state-of-the-art innovations prepare the innovator to adapt into the future technical changes better. This also suggests that companies should involve in development themselves and avoid the "not-invented-here" trap.
4. *Secrecy.* Many studies have shown that secrecy is relatively the least useful, though also possible mean for appropriation.

The little empirical research available shows that lead time, continuous improvement and skilled personnel have been in practice more important appropriation tools than copyrights and patents.¹⁹⁷

However, while legal protection may be a weak appropriation tool, copyrights and patents also open up possibilities to appropriate returns from the complements of innovation.¹⁹⁸ For example tacit knowledge required to use the innovation can't be protected directly by legal means but can be used as a sales argument for the software itself.¹⁹⁹ It must be noted, that patents may work against these market based incentives for software innovation. While with copyright different methods were complementary, in patents, they are more clearly alternatives.

3.3.5 An Open Innovation Model

New models that propose courageous outsourcing and free flow of ideas over company boundaries have questioned the traditional innovation management behind the closed doors of a company. Already Coase noted that if communication technology drives the costs of outsourcing

¹⁹⁶ E.g. Liebowitz and Margolis (2001) argue that Microsoft gained its dominant position in office software markets (text processing and spreadsheets) not because they were first to market or had intellectual property advantages but because their software was technically superior

¹⁹⁷ Tormsi (1998), p. 109-110. Levin et al (1987)

¹⁹⁸ See e.g. Teece (2000), p. pp. 115-

¹⁹⁹ Arora et al (2004), pp. 115-117

through the markets down more than organizational costs, it is arguably rational to reduce the size of the company's internal activities.²⁰⁰ Now, just because the acquisition and management of exclusive rights to the results of innovations is costly and, on the other hand, communication and voluntary cooperation costs on the markets have become down, it makes sense to consider an open model for most innovative activities.

Chesbrough presents one possible model:

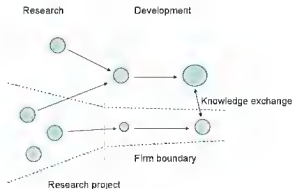


Figure 10. An open innovation model.²⁰¹

Roughly put, there are typically many research projects only a few of which ever enter the technological development stage. Cooperation and free exchange of research information across company boundaries may lead to new outside development projects, which combine innovatively the existing knowledge base. Later on, the company may have an advantage (based on prior merits to help launching the project) to benefit from

²⁰⁰ Coase (1937)

²⁰¹ Chesbrough (2003), p. 189

such external projects for example through informal knowledge exchange and licensing.²⁰²

This kind of open innovation model has obvious similarities to the economics of science.²⁰³ Informal disclosure of new information, exchange of research workers and the culture of cooperation may optimally produce new innovations more efficiently than closed technology projects. It is also well argued that an open source development model closely reminds that of the scientific research.²⁰⁴

We can find many economic reasons for profit-seeking companies to participate in basic research and vice versa. For example Foray has proposed that it makes economic sense to freely reveal research and development information if:²⁰⁵

- *Reward systems* address the issues of free copying and distribution. This comes back to alternative compensation mechanisms and innovation appropriation discussed above.
- Companies are required to use *reciprocity obligations* to participate in development cooperation
- Companies are interested in improving the *average performance of the industry*. For the software industry e.g. the development of more secure Internet infrastructure could be one area.
- Companies have an *open source strategy*, which implies that others have no incentives to commercialize inventions through proprietary means

A difficult question is how open innovation affects proprietary software development models in the software industry. One can argue that proprietary models won't disappear overnight even though companies would adopt open innovation models. Also proprietary licensing models can be

²⁰² In the case of open source licensing terms, every company is theoretically in the same line though. However, the initial developer may still have the advantages of leadership, tacit knowledge and informal relations within the project.

²⁰³ See Dasgupta and David (1994) for an overview of the differences and synergies between "The Republic of Science" and "The Realm of Technology"

²⁰⁴ See e.g. Kelly (2001)

²⁰⁵ Foray (2004), pp. 179-181

adapted to allow knowledge sharing and exchange to some limited extent without losing development control and ownership over copyrights and patents.

3.4 Competition Policy and the Limits of Exclusive Rights

Private property rights, intellectual property rights part of them, are commonly seen as a necessary condition for the creation of free markets. For instance North belongs to intellectual property optimists: he links them to technological change and defends the idea that only well-defined property rights to innovation can guarantee market-driven continuous economic development.²⁰⁶

However, also state intervention to intellectual property markets can be justified in economic terms. Historically, states had interest to censor the printing press and intellectual property rights developed in many jurisdictions as part of censorship laws. In modern times, state interest has shifted from censorship to guarantee the efficient functioning of free markets. The interest is now in the intersection between competition policy and intellectual property. It has turned out that software industry has been specifically vulnerable to market failures.

As noted earlier, software and computer industries have presented many textbook examples of inefficient monopoly behavior. Some of the history's biggest anti-trust lawsuits have been raised against computer and software manufacturers. Until the 1990s, IBM practically dominated the software industry for decades. IBM was considered for anti-trust claims in the United States starting from the late 1960s for bundling software with hardware. At that time IBM made it impossible for any company to sell software since IBM gave software for free with their industry standard hardware.

In the 1990s, when Microsoft had grown to be the monopoly software system on personal computers, the United States Department of Justice filed several anti-trust suits against the company demand-

²⁰⁶ North (1981), p. 162-166

ing among others that Microsoft should release some of their software in a non-exclusive way, possibly open source. Also the European Union has investigated Microsoft's practices. The company has been accused of e.g. bundling web browser and multimedia player software to its de facto operating system standard efficiently eroding the market of small companies offering competing software products.²⁰⁷

The basis of the monopoly problem is in the economic laws of networks. Winners are winning more. Intellectual property rights can further contribute to their monopoly power: an incumbent company may have the power to exclude others from markets through the execution of copyrights and patents. Such arguments have been repeated for example in the Microsoft case. However, the same economic laws of networks and intellectual property may also work for increasing competition. As Posner reminds, "competition to obtain monopoly is an important form of competition".²⁰⁸

Many economists argue that monopoly situations in the software industry are inefficient for the society at large. Poorer quality software will be produced because of decreased competition. Innovation will be stifled. However, Liebowitz and Margolis have argued that specifically Microsoft's monopoly position in office applications is not the result of lock-in, network effects and predatory pricing but simply because they have technically better products. When Microsoft was in the markets, the prices dropped in fact faster than in those markets where Microsoft was not present. Hence, they claim that consumer's will never lock-in to inferior products.²⁰⁹

The impact of open source to the IPR competition policy debate is perhaps two-fold. First, there is little doubt that the growing popularity of open source software and associated open standards has once again in-

²⁰⁷ For an overview of the recent developments in Microsoft cases see e.g. The Economist (2004a). Microsoft suits have also highlighted the challenge – as with many other legal issues – of geographical reach of the applicable laws. The possibilities of even the US and EU to influence the behavior of the world's biggest computer and software companies are limited.

²⁰⁸ Posner (2002), p. 248

²⁰⁹ Liebowitz and Margolis (2001)

creased competition in the software markets. There is no software company that could dominate the Internet as the *de facto* communication platform. For example Microsoft's attempts to "extend and embrace" its proprietary standard extensions to the Internet (web pages and email) have so far been failures. Interestingly enough, Microsoft has already used the growing popularity of open source as a defensive argument in their competition lawsuits.²¹⁰

Second, and quite paradoxically, one can also argue that open source may decrease competition in the long term. If open source in the sense Free Software Foundation pushes it forward becomes standard in specific application domains, it may severely limit proprietary licensing possibilities and in the end take incumbent proprietary software companies out of business. For example SCO has used such arguments in its case against IBM: if Linux becomes a *de facto* Unix standard, there may be no markets for proprietary Unix licenses in the future.²¹¹

3.5 Summary: Economic Rationale of Open Licensing

There are at least three kind of economic lines of reasoning to support the idea of open rather than proprietary software licensing. If we view software as:

1. A *copyrighted work*, then strong network effects and aim for market share support openness. The ability to use price discrimination supports the idea of more rights to certain groups of users. Further, in addition to licensing fees for copying and distribution, there are many alternative economically sustainable ways to generate income.

²¹⁰ See Microsoft's appeal brief in *US v. Microsoft* (2002) from November 27, 2000 where the company states that "[Linux supporters] could quickly expand their output to satisfy the entire demand for operating systems among OEMs if Microsoft were to stop innovating or begin charging a supracompetitive price for Windows."

²¹¹ See *SCO v. IBM* (2003), where SCO among others charged IBM on unfair competition since by supporting Linux "...IBM has engaged in a course of conduct that is intentionally and foreseeably calculated to undermine and/or destroy the economic value of the UNIX Software Code anywhere and everywhere in the world, and to undermine and/or destroy plaintiff's rights to fully exploit and benefit from its ownership rights in and to UNIX"

2. *A system product*, then compatibility is a strong argument for openness. However, many times open interfaces are enough for compatibility and open source code does not bring any additional benefits in this sense. Also a desire for lock-in and path-dependence speak for non-compatibility.
3. *An innovation*, then the cumulative and collaborative nature of inventions supports the idea of an open innovation process. Exclusive intellectual property protection is just one of the alternative means to appropriate innovation among lead-time, continuous improvement and skilled personnel. For example software patents have proved to be in practice rather an expensive strategic asset rather than an appropriation tool.

Overall, the economic theories of networks, copyright and patents emphasize a strategic approach towards a software licensing decision. Software companies need to consider additional environmental factors in intellectual property licensing and not limit their thinking into the traditional pricing issue. To simplify the picture, one could argue that much in the economic theory behind open source licensing is just another incarnation of the "Internet economics" of the late 1990s. For sure, unconditional exclusivity is doomed on the Internet. However, to succeed commercially one may not be able to "follow the free" until the bitter end.²¹²

²¹² For example Kelly (1998) famously popularized network economy into short lines such as "Follow the free", "To maximize innovation, maximize the fringes", and "Sooner or later closed systems have to open up, or die."

4 INTELLECTUAL PROPERTY AND ITS DISCONTENTS

This chapter discusses the evolution of legal and technical protection of software. Some commentators have criticized that the continuous expansion of different overlapping intellectual property laws over software has implied that the law is now substantially out of balance. We end the chapter by reviewing the evidence and discussing the possibilities of private balancing of intellectual property laws through open source.

4.1 Challenge of Software Protection

Since the emergence of computers after the Second World War, there was intensified academic discussion of the legal status of software. The main alternatives of copyright and patents were under closer study already in the 1960s.²¹³

4.1.1 Early Discussion and Practice

First patents. The first form of intellectual property explicitly granted to software was probably patent.²¹⁴ In Germany, two computer scientists patented what may constitute a software algorithm as early as in 1957.²¹⁵ In the United States, there was also increasing pressure towards the Patent Office to start accepting software patents.²¹⁶ Applied Data Research's

²¹³ See e.g. Puckett (1966) for an overview of the early discussion on both software copyright and patents in the United States and Koktvedgaard (1968) for discussion in Europe

²¹⁴ To be precise, software can't be (legally) patented. Only inventions can be patented. If an invention is implemented in software, however, one can speak of software patents. Thus, in legal jargon, the discussion is usually about "computer-implemented inventions" when the issue at hand is essentially software patents

²¹⁵ German patent (DE 1,094,019) was filed March 30th, 1957. It was also patented in at least the United States (US 3,047,228), France (FR 1,204,424), and England (GB 892,098). The invention is described in Samuelson and Baner (1960). Johnsen (1969) points out that there were a number of early German patents (DE 218,524, DE 255,921, DE 314,001, and DE 314,001) granted for mechanical calculators.

²¹⁶ Hirsch (1966) reported (p. 79) from a US Patent Office's public hearing on the issue that "so many answers were volunteered that it will probably be awhile, if ever, before program patents are granted". The magazine also explained that the issue of software patentability had created "a sometimes acrimonious debate that has been dragging on for at least 10 years".

(ADR) became the first company, which was granted a software patent in 1968 with commercial importance.²¹⁷

Other companies followed ADR's lead.²¹⁸ Another early commercial patent applicant was Whitlow Computer Systems, which was able to patent a hidden sorting feature in System/360.²¹⁹ Whitlow also competed with IBM's bundled software. Company founder Duane Whitlow later explained:

"I knew it was patentable because Marty Goetz had started the ball rolling in that direction. We had found one instruction in the System/360 computer that made a dramatic improvement in performance. Although there were significant other aspects of the sort that are included in the patent, I knew this aspect was a breakthrough and anchored the patentable methodology."²²⁰

The early practice of patenting came to a temporary halt in the 1970s. Neither courts nor patent offices accepted patenting as the legal form to protect software during the decade.²²¹ When personal computers were introduced and become popular in the late 1970's, the volume of developed software increased enormously and new firms entered the industry. Patenting clearly wasn't an option in such a rapid restructuring stage.

²¹⁷ Martin A. Goetz, the head of Proprietary Software Division of ADR, was named as the inventor of US Patent 3,380,029 - "Sorting System", April 23, 1968. For more background see description of ADR's Software Products Division Records in ADR (2003). ADR also sued IBM (not related to the patent) for mischaracterizing the features of its bundled flowcharting program and joined the so-called antitrust suit. The background was that ADR competed with IBM's bundled flowchart software on System/360.

²¹⁸ It should be noted that the opinions on software patentability were mixed within the industry. For example IBM remained opposed to software patents although it could have applied them easily. See e.g. Hirsch (1968).

²¹⁹ U.S. Patent No. 4,210,961 - also titled "Sorting System" of Duane L. Whitlow and Azra Sasson, was finally issued July 1, 1980.

²²⁰ Bude (2002).

²²¹ US Supreme Court cases *Gottschalk v. Benson* (1972) and *Parker v. Flook* (1978), although carefully limiting the decision to the particular facts of the case, denied software from patentability in the United States. European Patent Convention was signed in 1973, which excluded software from patentability in Europe. Also EPO's 1978 Patent Examination Guidelines confirmed the rule.

Copyright. While patents dominated the industry discussion, the first US copyright registration for a software product was recorded already in 1964.²²² The need to clarify the legal status of software continued to grow especially as software mass markets were about to form towards the late 1970s. Companies started to develop “shrink-wrap” contracts into software packages because it wasn’t clear if software copyrights to non-literal object code were valid at all.²²³

The problems of copyright were illustrated in the case *Data Cash Systems v. JS&A Group* a company lost a copyright suit against a competitor that sold its computer game without authorization.²²⁴ The competitor had literally copied object code from a physical game-module, then repackaged and commercially distributed it. The lower court decided in 1979 that copyright did not cover “object code”. Further, appeals court stated that the game wasn’t even under copyright since it lacked proper copyright notice; there was one in the manual but not in the game module itself.

Sceptical practise. Despite the heated discussion, many corporate software companies were not that worried about the fuzzy status of legal protection of software. A good example is the conclusion from an industry study in the 1970s where over a hundred software companies answered questions about the legal context of software licensing:

“Many of the firms surveyed were not greatly concerned with legal protection of software; many chose not to answer the question on preferred mode of legal protection. Those who did answer displayed a strong preference for contractual restraint through trade secrecy over either patent or copyright. ... only a small minority (4 percent) of respondents reported having abandoned the development of a program for lack of protection.”²²⁵

²²² Band and Katoh (1995), p. 71

²²³ See e.g. Tyler (1986) pp. 11- for coverage of the early discussion

²²⁴ *Data Cash Systems v. JS&A Group* (1980).

²²⁵ Muller et al (1978), part of CONTU

Thus, the early interest in the intellectual property protection of software was largely academic and bureaucratic. There wasn't any intellectual property licensing business in software at that time.

4.1.2 WIPO's Proposal

World Intellectual Property Organisation, founded in 1967, started to prepare provisions for the international protection of computer programs in the early 1970s.²²⁶ The work was completed in 1978. Instead of proposing direct amendments to copyright or patent treaties, WIPO drafted a model law based on a *sui generis* approach. The motivation was that software required specific kind of intellectual property regulation, which didn't adequately fit into either copyright or patent laws.²²⁷

Substantially, the model provisions stated that:

- New regulations would only complement copyright and patent laws and not replace any intellectual property protection already given to software
- A copyright approach in general should be taken regarding the protected rights (covering only literal copying but no independent creation)
- Following the principles of patent law, also the commercial use of software should be protected whether involving copying or not
- Also following the principles of patent law, the term of protection should be limited to twenty years
- An optional registration system should be considered to encourage the dissemination and disclosure of software

²²⁶ See WIPO (1971) for the first international meeting and agenda setting. IBM had already proposed a *sui generis* approach to the US Patent Office a decade earlier. See Galbi (1970) for details and also Husch (1969) indicating for an initially positive reaction to IBM's proposal by the US Patent Office. In Europe e.g. Koktvedgaard (1968) supported *sui generis* approach over patents and copyright.

²²⁷ The model law approach was taken mainly because it was felt that the regulation of computer programs should first develop and stabilize to some extent before the rules could be harmonized. See WIPO (1978).

The provisions further explained in a general utilitarian spirit that:²²⁸

"The primary purpose of the protection granted is not to allow proprietors to profit from a period of exclusive rights as a reward for the creation and disclosure of computer software, but simply to encourage creation and dissemination of computer software and to prevent the misappropriation of the results of another's valuable work..."

Despite its well-thought intentions the model provisions were not implemented into national legislations as such. A further attempt to prepare a new international treaty on the legal protection of computer programs was dropped in 1983.²²⁹ Instead, copyright laws were simply amended to include software as another new category among other literary and artistic works. The proposed limited term and optional registration system didn't get enough support in the international intellectual property regulation of software.²³⁰

4.2 Copyright and its Limits

During the 1980s copyright laws across the world were amended to include computer programs as another category of protected works. The so-called interoperability debate then defined the legal rights of software users in more detail. One can argue that software copyright is as of today a settled area of law without any radical legal development in sight.

4.2.1 Software Enters Copyright Law

A decisive point towards copyright came when the US National Commission on New Technological Uses of Copyrighted Works (CONTU) recommended in its 1978 report to apply only copyright to software:

²²⁸ WIPO (1978), p. 9.

²²⁹ WIPO (1983).

²³⁰ For example in a 1985 meeting, a WIPO Expert Group continued to discuss the term problem but didn't even mention the idea of a registration system. See WIPO (1985).

"The new copyright law should be amended 1) to make it explicit that computer programs, to the extent that they embody an author's original creation, are proper subject matter of copyright; 2) to apply to all computer uses of copyrighted programs by the deletion of the present Section 117; and 3) to assure that rightful possessors of copies of computer programs can use or adapt these copies for their use."

United States was subsequently the first country to introduce a specific Software Copyright Act in 1980; after its enactment, the unauthorized copying, distribution and modification of computer programs has been illegal as copyright infringement. Other countries followed and soon copyright was universally preferred to other forms of intellectual property. For example, a 1985 WIPO meeting noted that:²³¹

"...a great number of participants developed arguments in favor of copyright protection of computer programs; patentability of computer programs *per se* had been ruled out under the law of virtually every country... copyright, in its development, had proved to be flexible enough to extend to works of a technical nature."

Since the late 1970s, as software mass markets emerged, also software (shrink-wrap) licenses started to explicitly mention copyright and most companies relied on copyright as the main legal mean to backup the licensing model.

4.2.2 Interoperability Debate

Issue and arguments. The limits of software copyright soon became tested in the so-called interoperability debate. The question was whether non-literal elements such as interfaces were copyrightable and whether reverse engineering should be allowed. Many leading software products in

²³¹ WIPO (1985), p. 147

the 1980s had interoperable interfaces and the custom of reverse engineering was in most cases considered perfectly legal

For example, MS-DOS was designed to be interoperable to some extent with at the time popular microcomputer operating system CP/M. MS-DOS was actually written because there was a demand for a simple operating system running on the new Intel 8086 processors and Digital Research, the developer of CP/M, was still working on CP/M-86 with no information on when it would be available. Thus, a little company called Seattle Software Works decided to develop one in 1981. The commands and programming interface of MS-DOS were designed to be quite similar with CP/M and it was also possible to translate CP/M applications to run in MS-DOS.²³² Digital Research never initiated any legal action against the developers of MS-DOS nor Microsoft.²³³

Band and Katoh identified three lobbying groups in the interoperability debate:²³⁴

1. *Ultraprotectionists* who believed that software is copyrightable and both interfaces and reverse engineering should be protected by copyright. Their arguments highlighted e.g. a need for logical coherence in copyright law, *ex ante* incentive theory and the central role of software industry in the US national economy.
2. Those who argued for a *balanced view*, which accepted software copyright in general but was against reverse engineering and inter-

²³² However, in practice the interoperability didn't always mean that much because of the rapid technological development. Tim Paterson, the author of first versions of MS-DOS, explained "After making a goal of the 'translation compatibility', and the claims of 'rip-off' that resulted because I used the CP/M manuals, it turned out the this feature was almost never used. Instead of blindly porting old 8-bit applications, the software developers rewrote them or started over to take advantage of 16-bit capabilities (like much more memory)." Tim Paterson's email to the author on 8 June 2004.

²³³ The famous history of Microsoft selling Seattle's new operating system – at the time called Q-DOS (Quick and Dirty Disk Operating System) – in the front of Digital Research is another story. Digital Research eventually published their CP/M 86, later renamed as DR-DOS, in 1983. DR-DOS was never able to capture relevant market share from MS-DOS.

²³⁴ Band and Katoh (1995)

face protection in particular. The basic economic argument went that if copyright would be given to an interface of a *de facto* standard software product, then the monopoly power would also extend to the markets of compatible software. In general, copyright to an interface would cause more incompatibility in networks and less social benefits.²³⁵

3. Advocates of *minimal copyright* who basically went against the whole idea of software copyright in side with reverse engineering and interface protection. This group was lead by Richard Stallman and they relied on ideological and ethical arguments. Stallman wasn't very influential in the actual debate but rather served as an easy target for ultraprotectionists who tried to bundle him together with the advocates of the balanced view.

Europe. In Europe the debate centered on the hearing process of the proposed *software copyright directive*.²³⁶ There were two more or less equally strong parties fighting against each other – a clear contrast to the US where the supporters of interoperability had significantly less industry backup. In Europe, a number of dominant US companies of that era (Microsoft, IBM, Apple, Lotus, etc.) established ultraprotectionist Software Action Group for Europe (SAGE), which was aiming to get as stringent law as possible to curtail the European competition. They were lobbying to add user interfaces under the scope of copyright and, perhaps more importantly, trying to ban reverse engineering altogether. To counter this threat, mostly European companies (Amstrad, Bull, Olivetti and Fujitsu from Japan) formed European Committee for Interoperable Systems (ECIS), which advocated a more balanced view. The latter group also received some mixed support from the academic community.²³⁷

²³⁵ See e.g. Menell (1989). For a comprehensive list of the arguments used by different parties see Band and Katoh (1995), pp. 99–101.

²³⁶ The initial intention of the directive was to merely harmonize copyright protection of computer programs across Europe. In the end, the directive provided a forum to define the limits in the substance of software copyright.

²³⁷ See e.g. Dreier (1991), in footnote 3, listing over 10 articles on the interoperability issue published in one particular journal (European Intellectual Property Review) during 1989–1991.

In the end, the EU Commission sided with SAGE and made a proposal, which would have made it nearly impossible to create interoperable software. This wasn't the end of the case, however, since the European Parliament chose to support ECIS instead. Parliament adopted a substantial set of amendments to the proposed directive, including three key amendments dealing with the interface and reverse engineering issues. In the final directive text as accepted in 1991, interfaces were not under copyright and reverse engineering for interoperability was allowed.

United States. In the US, the debate was solved in the courtrooms right after the software directive was accepted in Europe. The first major case was *Computer Associates v. Altai*, decided in 1992, where the question was about a component that allowed software interoperability to different hardware systems. Altai's programmer had originally copied some 30% of Computer Associate's source code but when Altai became aware of the copying, they rewrote it. Computer Associates went on to argue that also the rewritten component infringed their copyright.

Both parties received support from other companies and different industry lobby groups. Large IT enterprises of the time including IBM, DEC, Microsoft, Lotus, WordPerfect and Apple supported an ultraprotectionist view. Their lobby group called Software Publishing Association (SPA) argued for Computer Associates. SPA's position wasn't united, however. Borland, Novell and former SPA member Compaq explicitly disagreed with their position. American Committee for Interoperable Systems (ACIS), which included less known companies such as Storage Technology, AT&T Global Information Solutions, Amdahl, and Broderbund Software, further supported Altai and had a more balanced view.²³⁸

In *Computer Associates v. Altai* the court rejected an earlier judgment, which stated that non-literal elements such as structure, sequence and organization of a computer program would be generally under copyright.²³⁹ Instead, the court went on to argue that copyright should not cover areas

²³⁸ Band and Katoh (1995), p. 102., go through these different groups and their intentions

²³⁹ In the earlier case *Whelan v. Jaslow* (1986) a court had stated that "... copyright protection of computer programs may extend beyond the programs' literal code to their structure, sequence, and organization"

where ideas merge into expression. It stated that programmers have less design choice in:

“(1) mechanical specifications of the computer on which a particular program is intended to run; (2) compatibility requirements of other programs with which a program is designed to operate in conjunction; (3) computer manufacturer’s design standards; (4) demands of the industry being served; and (5) widely accepted programming practices within the computer industry”.²⁴⁹

Thus, the court essentially stated that interoperability is not under copyright. Another central case was *Lotus v. Borland*, decided in 1995, which ended in the favor of Borland and also against interface copyright. Lotus had claimed that Borland infringed the copyright to Lotus 1-2-3 spreadsheet by copying most of the structure of its menu system. At first instance, the decision was in favor of Lotus but after a complaint, a federal circuit ruled in favor of Borland. The court noted that the question was about a “method of operation” and stated: “the ‘expressive’ choices of what to name the command terms and how to arrange them do not magically change the uncopyrightable menu command hierarchy into copyrightable subject matter” – US Supreme Court eventually took the case but was unable to make a decision: the vote was split 4-4 and the earlier decision became final.

4.2.3 Current Extent of Software Copyright

Copyrightability. It is now well founded that copyright covers any original computer program as a literal work whether it is in source code or object code form. For example, according to software copyright directive, “Member States shall protect computer programs, by copyright, as literary works within the meaning of the Berne Convention for the Protection of Literary and Artistic Works”. Only original practical expression is under copyright; non-literal elements, ideas and program functionality fall out-

²⁴⁹ See *Computer Associates v. Altai* (1992) and *Band and Katoh* (1995), pp. 125-126

side the copyright protection. Otherwise, copyright covers the program as a whole and thus builds a natural legal basis for licensing.

Copying and distribution rights. The substance of copyright is to give the author an exclusive right for the copying and distribution of the work.²⁴¹ Within the context of computer programs, it hasn't been usually productive to try to define any distinction between copying and distribution since distribution usually involves copying (thus distribution being a subcategory of copying).

However, many open source licenses do draw the line. They allow copying but may restrict distribution. This is because copying may be in certain cases necessary for the using of software and open source does not specifically restrict the mere running of software. Moreover, it must be stressed that *copyright does not cover the use of programs in general*.

Laws typically have a few relaxations to the exclusive rights of copying and distribution. Backup copies and necessary runtime copies are usually allowed. Also, though much more controversially, it may be legal to distribute a legally obtained copy of the software to a new user according to the *first sale doctrine*.²⁴²

Modification right. The right to modify a program is perhaps the most central and controversial right in many open source licenses.²⁴³ Namely some licenses place restriction not on the copying or distribution of the program as such but merely to the (distribution of) modifications.

The general rule is that whoever wants to modify a work needs the permission of the copyright holder. In the case of software, the modifier and the original authors typically hold the right to a modified source code

²⁴¹ To be exact, copyright laws speak of the rights of reproduction (copying) and making available to public (distribution).

²⁴² It is often noted that the *first sale doctrine* wouldn't apply to digital works since they are only licensed and no "copy" is sold when the program is distributed, however, some courts have considered the redistribution of original copies legal on the condition that the user does not keep any copy of the software. See e.g. Finnish Supreme Court decision KKO 2003 88 referring to the actual nature of (shrink wrapped) software licensing and the principle of free movement of goods.

²⁴³ Since modification right (in the US, *derivative works doctrine*) and its interpretation is so central in open source licenses, the right and its application is analyzed further below in section 5.3

jointly.²⁴⁴ In open source development, the crucial issue usually is, whether a given method of using existing programs as the basis of a new one should be counted as modification of the existing ones (and subject to the control of the original author) or the creation of completely separate new work.

Moral rights. Finally, copyright includes so-called moral components the most important and universally accepted being attribution and reputation.²⁴⁵ Attribution means that the author has the right of recognition of his authorship. Thus, for example the removal of author name from the work would be an infringement of the attribution right.

Reputation means, roughly put, that the work can not be modified, altered or used in any ways that would harm the reputation of the author.²⁴⁶ For example an intentionally low quality modification of popular open source software with many programming errors added could infringe this right.²⁴⁷

Liability. Although copyright liability rules have not been harmonized to the extent the substantial law in international treaties, the liability standard is uniformly *strict liability*. This means that the copyright holder has a cause for action against any infringer regardless of whether the infringement has been willful or not, and even whether it has been made by a business or by individuals. Willfulness and economic interest may only affect the amount of damages and compensation awarded to the copyright holder.²⁴⁸

²⁴⁴ To be exact, a translator may be given an exclusive right to his translation. However, to translate the work in the first place and later to market the translation, he needs the permission of the original author.

²⁴⁵ Legal scholars usually speak of rights of *paternity* (attribution) and *integrity* (reputation). See e.g. Goldstein (2001), pp. 283–. Moral rights are codified in international copyright treaties (e.g. Berne Convention etc.) and also in most national copyright laws a notable exceptions being the United States.

²⁴⁶ Copyright also knows exceptions to exclusive rights such as parody, which may conflict with this rule.

²⁴⁷ Modification is also subject to the modification right in copyright mentioned above. Naturally, such a malicious modification could be held e.g. as unfair competition and thus be against other laws too.

²⁴⁸ For example in Finland the Supreme Court has settled that the minimum compensation for a software copyright infringement is the “standard price of a license” (KKO 1998:91). If the act is

4.3 The Return of Patents

Not everyone accepted after interoperability debate that the legal protection of software was a settled area of law; copyright had only showed its limits. Some academics proposed the reconsideration of the *sui generis* approach but it never realized into actual policy proposals.²⁴⁸ Instead, the increasing use of patents rose to dominate the policy debate on software intellectual property.

4.3.1 United States Leads

It is commonly believed that the US Supreme Court case *Diamond v. Diehr* decided in 1981 was the starting point of increasing software patenting in the United States. In the case, the court stated that:²⁴⁹

“A claim drawn to subject matter otherwise statutory does not become nonstatutory simply because it uses a mathematical formula, computer program, or digital computer.”

As we noted, there were a number of software patents granted prior this particular case but earlier court cases had not held them valid. After *Diamond* case, software was thought more generally to be patentable although the case left some interpretation issues open. Obviously, if the invention consisted only of a mathematical algorithm it wasn't still patentable. Later court cases have however pushed the abstract interpretation problems in side favoring more extensive patentability of software.²⁵⁰ Thus, in 1996, the USPTO published more liberal *guidelines* for the exami-

willful, the compensation may be doubled and additional damages added on the top. In the US, the amount of compensation for software copyright infringement depends also on whether the infringer has acted willfully and, in addition, whether the copyright has been registered.

²⁴⁸ Samnelson et al (1994) being perhaps the most influential.

²⁴⁹ *Diamond v. Diehr* (1981).

²⁵⁰ The most important being perhaps in *Re Alappat* (1994), which forced USPTO to write new guidelines for the examination of software patents. Jaffe and Lerner (2004) argue that increasing patenting in the US (not limited to software patents) was also helped by administrative procedure in 1982 to establish a common instance (Court of Appeals of the Federal Circuit), which held patents more often valid. In addition, USPTO had an incentive to grant more patents after it was ordered in the early 1990s to run on its fees instead of taxes.

nation and granting of software patents. The guidelines essentially stated that the utility of the invention as a whole should be examined in the first place before a detailed assessment of whether the invention is solely based on an abstract mathematical algorithm.²⁵²

“Office personnel will no longer begin examination by determining if a claim recites a ‘mathematical algorithm.’ Rather, they will review the complete specification, including the detailed description of the invention, any specific embodiments that have been disclosed, the claims and any specific utilities that have been asserted for the invention.”

Software patentability received mixed response from the US software industry. In a public hearing in 1994, companies such as Oracle, Adobe and Autodesk opposed patentability while IBM, Microsoft or SUN were in favor either unconditionally or proposing improvements to increase the quality of patents.²⁵³ Soon afterwards towards the end of 1990s, when more extensive software patentability had become the norm, companies went on to adopt more favorable public positions towards patents and started to gather patent portfolios of their own.²⁵⁴

4.3.2 Europe Follows

European Patent Convention and its interpretation practice. At the other side of the ocean, software patents remained in the periphery for longer. European Patent Convention, signed in 1973, stated that computer programs were not patentable “as such”. On the one hand there was a need for legal clarity but on the other hand it was considered important that the development of a new industry would not be hurt by too strict

²⁵² US Software Patent Guidelines (1996)

²⁵³ See e.g. Ham (1998) for a collection of testimonies and other public policy material from that period

²⁵⁴ See section 6.1 for statistical evidence

definitions and a categorical denial of software patents.²⁵⁵ European Patent Office's first examination guidelines from 1978 didn't rule software patents out but stated still somewhat cautiously:

"A computer program may take various forms, e.g. an algorithm, a flow-chart or a series of coded instructions which can be recorded on a tape or other machine-readable record-medium, and can be regarded as a particular case of either a mathematical method ... or a presentation or information ... If the contribution to the known art resides solely in a computer program then the subject matter is not patentable in whatever manner it may be presented in the claims. For example, a claim to a computer characterised by having the particular program stored in its memory or to a process for operating a computer under control of the program would be as objectionable as a claim to the program per se or the program when recorded on magnetic tape."

Following the policy changes in the United States during the 1980s, European Patent Office started to accept software patent applications more liberally. In 1986, much like in *Diamond v. Diehr*, EPO decided that an industrial method implemented in a computer program and the necessary hardware was patentable even though the innovation resided solely in the computer program.²⁵⁶ In 1998, much like in *In Re Alappat*, EPO went further noting that:²⁵⁷

"...a patent may be granted not only in the case of an invention where a piece of software manages, by means of a computer, an in-

²⁵⁵ On the negotiation history, see Berensford (2000) p. 18-20 and Münich Diplomatic Conference (1973). Head of the committee for patentability criteria noted that "European Patent Office would simply have to be relied upon subsequently to interpret this expression unequivocally"

²⁵⁶ EPO T 0208/84 – VICOM: "A claim directed to a technical process which process is carried out under the control of a program (whether by means of hardware or software), cannot be regarded as relating to a computer program as such. A claim which can be considered as being directed to a computer set up to operate in accordance with a specified program (whether by means of hardware or software) for controlling or carrying out a technical process cannot be regarded as relating to a computer program as such."

²⁵⁷ EPO T 1173/97 – IBM

dustrial process or the working of a piece of machinery, but in every case where a program for a computer is the only means, or one of the necessary means, of obtaining a technical effect ... for instance ... achieved by the internal functioning of a computer itself under the influence of said program."

In other words, a required "technical effect" for any computer-implemented invention didn't require any hardware environment. After this decision, it isn't difficult to think that software "as such" is indeed patentable contrary to what EPC literally says. At least it is clear that in practice software patents have never been banned in Europe. In addition, the practice of granting patents has not differed fundamentally from the US.

Recent political debate. At the end of 2000, a diplomatic conference considered whether it was time to codify EPO's practice and remove the "as such" language from EPC. To the surprise of many, the meeting did not remove the language. Independent programmers, some small software companies and open source enthusiasts had organized to lobby against the change and in the end convinced the majority of EPC member states to reject the proposal.²⁵⁸

Meanwhile, the EU Commission had started preparations to harmonize national legislations regarding software patents.²⁵⁹ After a consultation period, Commission published a directive draft in 2002, where it wanted to clarify that software was indeed clearly, but also limitedly, patentable in Europe.²⁶⁰ An intensive public discussion followed including open source activists, individuals and some small software companies opposing software patentability altogether. Mainly large software and telecommunication companies, patent attorneys, and other intellectual property specialists supported practically unconditional software patentability. A number

²⁵⁸ See EPC (2000), p. 69 and e.g. Morrison & Foerster (2000) for an overview of the debate.

²⁵⁹ Green Paper (1997), pp. 16-17

²⁶⁰ As an important detail, the original directive proposal went against the IBM decision from 1999. It would have codified the patentability criteria largely according to the VICOM decision from 1986.

of studies were commissioned to examine the public policy benefits and costs of software patents.²⁶¹

In 2003, the opponents gained victory in the Parliament's first reading. Significant amendments were voted to the directive proposal. For example, it was clarified that any software operation aimed at interoperability never constitutes a patent infringement. Also, pure data processing would never constitute a patent infringement. It defined that "the processing, handling, and presentation of information do not belong to a technical field, even where technical devices are employed for such purposes". The wind changed quickly, however. Council of Ministers favored the pro-patent lobby in 2004. It practically erased all the amendments added by the parliament and further modified the directive to clearly accept EPO's practice from the 1990s. As of this writing, the directive proposal waits a second reading at the parliament. Changes are possible.

The European discussion has differed from the US in several respects. In the US the patentability critique is targeted towards the requirements of novelty and inventiveness – not towards a categorical limitation of software patentability ("as such" -language). In the long-term, issues in the actual use of patents such as competition policy, limitations on liability and patent claim interpretation have become more relevant than patentability criteria. Perhaps the European discussion will again follow the US and move on to the critique of the patent system as a whole after the directive process is finally complete.

4.3.3 International Policy

United States and different trade organization pressures have pushed countries all over the world to generally provide up-to-date patent laws providing, for example, legal protection to pharmaceutical innovations. Software has followed in the side. Critics have argued that software markets differ significantly from pharmaceuticals. Since software development is generally not that capital intensive, even companies from developing countries may be leading inventors.

²⁶¹ See section 6.2 for a more detailed discussion

This makes one ask what are the possibilities of each country to have a specific policy towards software patents? The Treaty on Trade Related Aspects of Intellectual Property Rights (TRIPS) signed in 1994 as part of the World Trade Organisation's (WTO) Uruguay Round negotiations, sets minimal rules for national patent laws. Article 27 of the treaty states:

"...patents shall be available for any inventions, whether products or processes, in all fields of technology, provided that they are new, involve an inventive step and are capable of industrial application"

It has been frequently debated whether article 27 requires that every country should accept software patents.²⁶² There seems to be nothing in the wording of the article, which would limit the patent protection of software. However, a detailed legal interpretation of the article isn't that simple. One can go the basics and ask how to define a software invention. Can software be invented at all? Are software inventions capable of "industrial application"?

Clearly, TRIPS treaty can be used as an argument for wider software patent protection and, still, it leaves enough room for sound economic policy discussion, which takes into account the interests of the local software industry and consumers. Both very wide and narrow patentability of software is possible. In the United States, as an extreme example, the field of patentable inventions has gradually extended from software to include even "business methods" and other more abstract ideas.

4.3.4 Current Extent of Software Patents

Patentability. Only inventions that are new, include an inventive step and have industrial application can be patented. In Europe, inventions must also meet the requirement of technical effect. As is usual with such

²⁶² See e.g. Foundation for Free Information Infrastructure (2004) questioning that TRIPS would imply software patents

legal definitions, the requirements for patentability have almost as many interpretations as there are readers.²⁶³

The US patent law does not mention software or computer programs. European Patent Convention instead does, but as noted above, in practice it doesn't categorically limit the patentability of software.²⁶⁴ It is commonly believed that the European Patent Office has accepted thousands of patents, which can be counted as "pure" software patents.²⁶⁵

Although patentability criteria has many similarities between the US and Europe, there are some differences. Therefore, it is relevant to note that US patents are not valid in Europe and vice versa. Thus, if software based inventions are more easily patentable in the United States, the effect of US patents must be taken into account only in the US markets. Also the implications of patents to open source development should be local.

Rights and exceptions. A valid patent gives the innovator an exclusive right to the commercial use of the invention. For example the making, marketing and selling of a product that includes or indirectly utilizes the patented invention is not allowed. In practice, then, patent covers much broader area of activities than copyright. The exact breath of a patent depends on the actual claims (or more exactly their interpretation) in the patent application.

Patent laws also typically include several exceptions for non-commercial or research use and prior use. Prior use exception means that an earlier innovator who hasn't applied for a patent can continue the use of the innovation without licensing concerns. However, expanding the commercial use of the innovation is not possible.

Liability. As with copyright, the liability standard for patent infringements is strict liability. One key difference however is that since patent rights apply only to commercial use of the invention, individual non-

²⁶³ "Official" interpretation guidance is given in USPTO's Examination Guidelines (1996) as well as in EPO's Examination Guidelines (2003)

²⁶⁴ Article 52 (c) of EPC defines that "schemes, rules and methods for performing mental acts, playing games or doing business, and programs for computers" are not to be counted as patentable inventions "as such"

²⁶⁵ E.g. Beresford (2000), p. v, and Stallman (1999b).

commercial infringement may be exempted from liability.²⁶⁶ Also the amount of compensation for infringement depends on whether the infringer has acted willfully.

4.4 Technical Protection

Practical enforcement problems of both copyright and patents have guaranteed a market for different technical self-protection systems. Despite many hopes and new initiatives, copy protections haven't offered any final solution to intellectual property enforcement.

4.4.1 Early Copy Protection Systems

Different technical ways to enforce usage restrictions on software were developed right after the early corporate software product markets started to form.²⁶⁷ Later, when software mass-markets evolved in the early 1980s, software producers started to use extensively different copy protection mechanisms to prevent unauthorized copying.²⁶⁸ Many early copy protection techniques were based on hardware tricks. For example, the popular spreadsheet program Lotus 1-2-3 had a following kind of system:

"The program required the user to initial the 123.EXE loader with owner information before the program would run. However, this initialization could only be performed on the original system disk. It turns out that the system disk contains a specially formatted track, which must be present for the initialization process to proceed (this

²⁶⁶ While it may sound intriguing with open source, this exception may not be that relevant in practice since open source explicitly does not restrict the commercial use of the software.

²⁶⁷ However, for example IBM didn't even take into use technical protection systems. See section 2.2.1.

²⁶⁸ Tylei (1986), p. 34, describes the sentiment in the early 1980s as "virtually every major firm has some form of copy protection, and some observes say that firms who do not protect their software are bringing poacy upon themselves". He also mentions interestingly that Apple representative, as contrast to many others, had commented that they can't sell a computer system "if it looks harsh or threatening".

track is completely trashed by the install program after the disk is initialized making it impossible to reverse the process).²⁶⁹

The copy protection systems didn't however solve the problem of copy-right piracy. Instead, software buyers became annoyed since copy protections compromised usability.²⁷⁰ For example, when hard disks became more popular and copy protections effectively prevented software installation on the hard disk. Copy protections also prevented backup copies.

"So copy-protection removal software was born. Its original intent was to let legal software owners make legal backup copies of their software. In a day when some word processor, spreadsheet, and other software cost US\$495 or more it was much too costly to just go and buy another copy if one or more disks went bad. And many refused to pay software companies up to US\$25 PER DISK for replacements. They would simply buy a less expensive product."²⁷¹

In the end, none of the software copy protection initiatives, were they based on hardware or software, proved successful enough to become standard. The role of copy protections was re-evaluated and many leading software companies chose not to use them at all.²⁷² As the markets for copy protection systems dried out well before the 1990s, the software industry had to start tolerating unauthorized copying to some extent

4.4.2 Anti-Circumvention Legislation

Legislation once again trailed behind the developments in the marketplace. A 1988 case had explicitly stated that copy protection systems did

²⁶⁹ See Tomboy (1998) for a collection of copy protection mechanisms (and easy cracks for them) used in Lotus 1-2-3 spreadsheet application during the 1980s.

²⁷⁰ Describingly, BYTE magazine readers nominated in 1986 as the software company of the year all those companies who dropped copy protections. See Poinelle (1987).

²⁷¹ See Rose (2004) for an introduction to copy-protection removal software.

²⁷² Poinelle (1988) reported that "as it happens, 1987 was the year the bottom fell out of the copy-protection market. A few publishers out there didn't get the word, but for the most part, copy protection of business software is history, and even publishers of games are going to an entirely different scheme."

not enjoy any special legal protection in the US.²⁷³ However, as quick as in 1991, EU's software directive came to add a new rule, which stated that the circumvention of technical copy protections is illegal.²⁷⁴ United States followed in 1998 with the implementation of Digital Millennium Copyright Act.²⁷⁵

Especially the latter legislation was connected with the arrival of the copy protection mania into the media industry.²⁷⁶ Relabeled as "digital rights management", copy protections were implemented to different media from physical CDs and DVDs to digitally distributed video and games. Sometimes these media copy protection systems have also been failures much like the software copy protections of the 1980s. A good example is the copy-protected music CDs from the early 2000s, which were in practice easily circumvented by virtually any computer user. Since those protection systems only made CD listening more difficult to music buyers, labels had to stop using them. DVD copy protections have lived longer although by now the system can be easily circumvented with generally available software.

4.4.3 Is Technical Protection Effective?

One central question common to all copy protection system is whose behavior they can exactly control. It seems that different user groups have

²⁷³ See *Vault Corp. v. Quind Software Ltd.* (1988), where the defendant manufactured a copy protection circumvention software called RAMKEY. The court held that "(1) Quind did not infringe Vault's exclusive right to reproduce — (2) Quind's advertisement and sale RAMKEY does not constitute contributory infringement, (3) RAMKEY does not constitute a derivative work — and (4) the provision in Vault's license agreement, which prohibits the decompilation or disassembly of its program, is unenforceable."

²⁷⁴ It must be noted, that the Software Directive's circumvention ban had many exceptions. If the circumvention had any lawful justification — such as back-up copies, research use, or the development of interoperable software — then the circumvention was obviously allowed.

²⁷⁵ In Europe, the extensive protection of technical copy protection systems for other works but software arrived with the 2001 Copyright Directive.

²⁷⁶ Difference between the history copy protection circumvention legislation on software and media industries also hints of the lobbying power of the respective industries. While the software industry has never had unanimous opinions on copy protections, the media industry has been particularly unified and also successfully furthering their goals in outlawing circumvention.

different capabilities and interests to circumvent the systems.²⁷⁷ Software-based systems work perhaps against what one could call *normal users*. But the *power users*, who are able to configure their computers and media devices outside manuals, will always be able to find cracks from the Internet. Legislation that prohibits the use and distribution of such crack programs has not had any significant effect for their availability. Finally, hardware-based solutions may raise the bar to crack the system even out of touch from the power users.²⁷⁸ But so far there have always been *professional crackers* who have in fact competed with each other who is the first to circumvent the latest copy protections systems were they based on hardware or software.²⁷⁹

Normal user	Power user	Professional cracker
No technical knowledge	Good technical knowledge	Excellent technical knowledge
Any protection system works	Software-based systems do not work	Can eventually crack any system

Table 6 User capabilities and technical copy protection systems

4.4.4 The Promise of Trusted Systems

Despite the unfortunate history, there remain those who believe that copy protections will come back and prevail one day. One argument is based on the idea of *trusted computing*, where every PC in the future would have a standardized hardware-based security system enabling among others copy protection. The argument is currently advanced by Trusted Computing Group (including IBM, HP, Microsoft, SUN, Sony, Intel and AMD) and strongly opposed by Internet activists campaigning for user rights and

²⁷⁷ The following is further developed from Scheer's (2001) taxonomy

²⁷⁸ Although for example power users of game consoles have been able to install additional circumventing hardware circuits to their systems

²⁷⁹ Reus (2001) presents an insightful economic analysis of the cracker profession. The "scene" consists of various groups competing on who first cracks and publishes the recipe for software-based copy protections

privacy.²⁵⁰ Another argument goes that in the future the increasing features of handhelds devices and game consoles with embedded software may significantly lower the market share of general purpose PCs.

A concern for open source developers is that if any of the trusted computing proposals would eventually become a standard, it would mean that users do not anymore have unlimited access to their system. It may be possible to build copy protection systems, which would be effective in normal and power user groups. Further, such a trusted system may require every software product to be authenticated by third parties who may not prefer open source.²⁵¹

4.5 Are Intellectual Property Laws Out of Balance?

One can argue that the continuous expansion of different overlapping intellectual property laws over software has implied that the law is now out of balance. But is it really so? Can open source balance the expansion trend?

4.5.1 Balancing Principle

Conflicting interests. One of the fundamental principles of intellectual property laws is the aim at balance between the interests of rights holders and the general public. On the one hand, absolute monopoly power with unlimited exclusive rights would arguably unnecessarily restrict access to culture and stifle innovation. On the other hand the existence of limited exclusive rights can be justified by utilitarian (incentive theory) and moral-philosophical (man's right to his personal creation) principles.

Public domain versus regulation. From a practical perspective, intellectual property protection can be thought to be arbitrary; in the case there were no copyright or patent laws, any information or knowledge would be free for anyone to replicate, build upon, and use. Also the very nature of

²⁵⁰ See e.g. Anderson (2003) for a critical overview and arguments against trusted computing initiatives.

²⁵¹ See Stallman's (2002) essay "Can you trust your computer?" where he argues that *treacherous computing* is a more adequate term to describe trusted computing initiatives

software as fully replicable digital information makes it impossible to control unless there is explicit legal regulation.

Therefore, any aspects or part of software products where the intellectual property regulation does not reach remain effectively free to use for any further purpose. Indeed, neither copyright nor patent or any other intellectual property right fully covers all aspects of software. For example copyright does not apply to those parts of source code with only one possible implementation, it does not cover functional aspects of a running software product and, in addition, copyright itself includes specific exceptions for particular uses of works otherwise under copyright. Patents are limited only to those parts of software, which implement an innovation – typically an algorithm in source code or other clearly separable subpart of the whole software product. Also patents have exceptions and so on.

4.5.2 Expansion Trend

Implicit balance. Historically, intellectual property laws have lagged behind technological development. So has it been with software too. As late as in the 1970s legal issues were rarely taken into account at software development even within the largest companies.²⁸² In the early corporate markets, when each user signed a separate contract and the possibilities for users to copy software and benefit from doing so were limited, contractually enforced trade secret law turned out to be the preferred legal form (if anything was needed) to protect software secrets. At that time it wasn't yet clear whether copyright and patents would apply to software at all. Nevertheless, it is important to note that the industry quite effectively achieved *the intent of the intellectual property* to create new works and innovations for the benefit of both the society and individual creators and innovators – without any explicit legal regulation.

Explicit regulation. Since the early 1980s, copyright law was finally explicitly revised to cover software products in detail. The economically very valuable software mass markets of the 1980s called for different means of

²⁸² National Research Council (1991), p. 5.

legal protection and the book and record publishing model based on copyright proved useful. Patents also returned into the spotlight in the 1990s when the limits of copyright became obvious. Now most of the big software companies have adopted a pro-patent policy and started to build portfolios of their own.

Table 7 below summarizes different protection methods used in software products during the industry history.

Era	Users	Computers	Software type	Software protection
1950s - 1960s	Corporations	Custom made	Corporate projects First products	Trade secrets Rare patents Nothing
1970s	Corporations Universities Hobbyists	Mainframes First PCs	Corporate products Internet infra Hobbyist projects	Trade secrets Nothing
1980s	Consumers Corporations Universities	PCs Mainframes	PC applications Corporate Products Internet infra	Copyright Technical protection Some patents
1990s	Whole society	All electronic devices	All kind for all users	Copyright More patents Technical protection

Table 7. Different forms of legal protection of software in the industry history.

Some of the major copyright and patent policy events during the industry history are summarized in the timeline below:

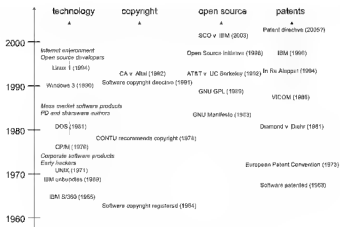


Figure 11. Evolution of software copyright and patents in the industry history

The legal development of software copyright and patents has been far from self-evident. The industry has witnessed difficult policy battles in almost all major legal developments from the interoperability exception in copyright to the recent debate about the applicability and coverage of software patents. If anything, it can be argued that the copyright basis of software is rather settled and balanced. One must admit, though, that patents are currently heatedly debated. While the role of patents is increasing the future remains still uncertain. Further, most of the alternative proposals to copyright and patents from *sui generis* protection to technical copy protection systems have not realized the hopes many have had.

Is the balance now endangered? Some commentators have argued that the expansion trend has endangered the balance of interests behind intel-

lectual property laws.²⁸³ The scope of different intellectual property rights has indeed continuously expanded to cover new valuable aspects of software thus strengthening the relative power of those who can acquire and use the rights. Any potential problems are further multiplied by the combined effect of different rights. Software is today covered not only by copyright and patents, but also trade secrets, technical protection, contracts and trademarks. The unfortunate outcome of the protection trend is that copyright and patents, among other rights, may in practice overlap.²⁸⁴

These overlaps can further contribute to the fragmentation problem and cause inefficient anti-commons lock-ups. A user needs may need to get a license from multiple right holders, not just one of them. Obviously licensing becomes more difficult and costly. Overlaps are also a major risk in open source licensing, as we will discuss later.²⁸⁵ A user may longer trust that licensed software covers all possible intellectual property rights just because it is very well possible that some unknown third party has for example a patent covering essential functionality.

4.5.3 Open Source as a Balancing Force?

Open source entered the legal policy development at the turn of the millennium. In theory, at least, open source offers noticeable balance to the effects of intellectual property rights. Open source strengthens the role of individual authors and software users over the traditional intellectual property right owner and proprietary licensor. This applies, however, only to the ways in which intellectual property rights are being used in practice.

Open source policy advocates have realized that the formal legal institutions are much harder to influence upon than informal social and ethical norms. The first major battle where advocates have had real influence is the software patents controversy in Europe where it is far from certain that

²⁸³ Of most notable critics, e.g. Lessig (2001) claims that "Rather than 'wait and see', the law has become the willing tool of those who would protect what they have against the innovation the net could promise". Diabos and Braithwaite (2002) further argue that the expansion trend in both patents and copyrights is leading the world towards "information feudalism" where multinationals exclusively own the culture and innovation.

²⁸⁴ As noted, copyright usually always covers the whole program and a patent may then cover certain patented functionality.

²⁸⁵ See section 6.1

their opposing arguments would prevail.²⁸⁶ Open source has brought again notable balance to the patenting discussion providing a credible counterargument to those who propose extensive US-style patentability. There is hope that also the law on software patents ultimately becomes settled and balanced as software copyright already largely is.

At a more general and global intellectual property policy level, World Intellectual Property Organisation (WIPO) is in the key role. It has been criticized of pushing in its agenda an *a priori* principle of "more protection is always better" without much economic analysis or reality checks.²⁸⁷ In fact, WIPO was about to set up a meeting to discuss open source and its impact on intellectual property already in 2003. However, largely due to pressure from BSA and USPTO the meeting was cancelled. An USPTO's official made an uneducated but revealing comment in public noting that:²⁸⁸

"To hold a meeting which has as its purpose to disclaim or waive such rights seems to us to be contrary to the goals of WIPO"

In 2004 consumer and civil society groups, campaigning for open source among others, were finally able to push alternative perspectives in the WIPO agenda from a new direction: allying with the interests of developing countries. Their proposal, which was accepted by WIPO for further examination, essentially concluded that.²⁸⁹

"A vision that promotes the absolute benefits of intellectual property protection without acknowledging public policy concerns undermines the very credibility of the IP system."

²⁸⁶ Europe's software patent debate is discussed in more detail in section 6.2.

²⁸⁷ For a book-length argument and evidence from different fields of intellectual property see Braithwaite and Drabos (2002).

²⁸⁸ Krim (2003).

²⁸⁹ WIPO Development Agenda Proposal (2004). It must be noted, though, that any agreed WIPO documents so far remain at the level of abstract diplomatic rhetoric without any direct reference to for example open source.

4.6 Concluding Remarks: An Open Perspective on Intellectual Property

As we saw, intellectual property laws have evolved to cover software products in finest details. Today, copyright, patents and technical protection systems are applied to govern the use of software products. But are these laws really necessary for licensing? This is a crucial but yet unanswered question. The fact is that software has been licensed even before there were any software copyright or patents in existence. In practice, then, intellectual property enforcement has not been a fatal problem for software licensing business.

One must be careful not to mix copyright and patents. One can argue that the function of patents in the software industry has not been to enable licensing in the first place. Patents give the right holders power to *exclude* others from using software. Copyright, instead, does not in practice give to its holder much power to exclude others since copyright covers only code and the code can be always rewritten to cover the same desired functionality. Nevertheless, copyright enables some *control over source code reuse*. Some open source licensing models essentially build on the possibility to control and enforce recycling.

The difference between the power to exclude and the option to control source code recycling is notable. The first, based on patents, applies to every software product even if they are independent creations. In this sense, patents are usually considered to go against the principles of open source since they can stop the creation of new independent works. The latter notion of controlled recycling, however, fits with open source principles. One may first require source code publication and then apply it to the direct derivatives of the original code. In this way, copyright enforces the principles of open source.

In fact, open source is developing copyright towards the original idea of *droit d'auteur*.²⁰⁰ The concept of a copy is dismissed, since in open source rhetoric the problem of copying belongs to the 16th century. Authors' rights, as they were originally developed in the 18th century France, went

²⁰⁰ See also Metzger and Jaeger (2002), pp. 94-95.

against the rights of the printing press owners who controlled the reprinting of copies. The commercial printing presses of the 20th century – software publishers, record labels and the like – reinvented the concept of copyright and the personal rights of the authors were pushed to the footnotes. One way to look at open source is to see it promoting the original ideals of authors' inalienable rights to control the integrity and paternity of their personal creations. At the same time open source also aims to undermine the role of patents. Thus, the impact of open source on intellectual property is not straightforward: while some aspects are clearly attacked, others are revitalized.

5 OPEN SOURCE LICENSES AS ALTERNATIVE GOVERNANCE MECHANISMS

This chapter discusses how different open source licenses build on intellectual property laws and reflect the lessons from the economic theory. Licenses are categorized and their functionality further analyzed. Open interpretation issues as well as implications to software business are identified. Finally, the chapter ends with a discussion on open content, on how the techniques of open source licenses have been adopted for use in other works of art than computer software.

5.1 Bargaining in the Shadow of Intellectual Property Law

5.1.1 What Makes a License Open Source?

Open Source Initiative formally accepts licenses that fulfill Open Source Definition as open source licenses. The definition requires essentially that a qualified license should allow:²⁹¹

- *Free use* meaning that any discriminating restrictions on e.g. commercial use, the number of users or hardware are not allowed²⁹²
- *Copying and distribution without any royalties* meaning that licensing fees are not a viable business model²⁹³
- *Modification without any royalties*.²⁹⁴ However, it is possible to include other conditions on modification such as the requirement to publish all modifications
- *Open and easily available source code* (but not necessarily free of charge), which is a practical requirement to do any modifications.²⁹⁵ Consequently, secrecy is not a possible mean to control development and innovation.

²⁹¹ For a somewhat similar summary of OSD, see Rosen (2004), pp. 9–11.

²⁹² See OSD, sections 5 and 6.

²⁹³ OSD, section 1.

²⁹⁴ OSD, section 3.

²⁹⁵ OSD, section 2.

These are essentially requirements based on the substance of copyright law. The basic logic is that the exclusive rights are reversed into non-exclusive. All major components of copyright – copying, distribution and modification – must be explicitly allowed in open source licenses. This contradicts strikingly the traditional thinking of intellectual property rights and licensing in the software industry.

It must be stressed that open source code is by no accounts anti-copyright. Licensing software with an open source license does not mean the software would be donated to the public domain. Open source software still has owners as long as copyright lasts. In fact, some open source licenses pose surprisingly strict obligations on the use of the modification right as will be discussed in this chapter. In addition, the moral components of copyright (mainly author attribution) are explicitly enforced.

5.1.2 What Is Not Required?

Regarding the content of typical software licenses, Open Source Definition is not that extensive. There are no requirements for many basic issues in other intellectual property laws than copyright. There are no requirements on license compatibility, warranties or formalities either.

Moral rights. Open Source Definition is silent on both attribution and reputation. No doubt, the copyright law in most countries does require for example that the author's name is not removed from a derivative work. But as was noted in the previous chapter, moral rights have not been codified in the US copyright law. Luckily, in practice every open source license requires attribution. As was argued at the end of the previous chapter, open source can be understood in fact to work largely based on the ideas of moral rights in copyright.²⁹⁶

Patent and other intellectual property licenses. As noted, Open Source Definition places essentially requirements on copyright but not on other intellectual property rights. The definition is silent on patents. Obviously,

²⁹⁶ See section 4.4

the requirements on royalty-free use, copying, distribution and modification imply that there can't be any patent licenses that require royalty payments. The same goes for trademarks. However, it seems to allow royalty collection from patents and trademarks targeted to others but the particular open source users of the software. For example, a licensor may license his patent without royalty to the open source users, but at the same time, collect royalties from the users of another proprietary software implementing the same patented invention.

Controlled rights ownership. Open source licenses do not require controlled rights ownership but allow distributed and fragmented rights. This also implies certain caveats as was discussed in the previous chapter.²⁹⁷ Thus, for example Free Software Foundation suggests authors who use their licenses to transfer the copyright to the foundation in the case there is need of license enforcement.

Warranties. The definition does not require anything on warranties either. This is actually rational, since the sales of both technical (error) and legal (indemnification) warranties can be an additional income source for commercial open source developers. Requiring a certain kind of minimal warranty would weaken the possibility of this business model and also frighten developers who can't accept any kind of liability for what they do. The lack of any warranties also highlights the possibility of liability caveats.

Compatibility. Open Source Definition sets only criteria to certify software licenses. It is not a standards specification, which would aim at interoperability and thus the concept of open source must be kept separate from open standards. In practice, sometimes open source code can't be combined with other open source code because the licenses of these two source codes are incompatible with each other. Admittedly, incompatibilities with licenses may result in less network effects and slow down innovation

²⁹⁷ See section 4.2.5

Formalities. Finally, the definition does not require that the licenses should be legally binding or enforceable. The validity of any provision in an open source license depends on its legal enforceability. The question of validity is especially important in countries other than the United States since most open source licenses have been written and legally analyzed from US perspective. Legal studies on the subject seem to conclude, however, that open source licenses should be enforceable in practice all over the world.²⁹⁸

5.1.3 Enforcing an Open Source Bargain

Legal theory. The definition does not say either whether the license should be based on solely copyright law or be written as a contract. There has been much debate on the benefits and drawbacks of bare copyright licenses versus contracts – the outcome perhaps being that contracts may need explicit acceptance from the part of the licensee while resting legally on a sounder basis.²⁹⁹

It has been argued for as long as software licenses are used that such license agreements may not be enforceable because the user may not consider and explicitly accept the license. However, many commentators of open source licenses argue that the licenses become indeed enforceable when the user distributes the work further – distribution is restricted by copyright until the license is accepted.³⁰⁰ Thus, the logic of the interpretation is that open source licenses are (mass market) contracts, which copyright law makes enforceable: by using any of the rights granted in copyright law the user must accept the license as a whole.

Para-legal practice. While it is possible to construct arguments for a strong legal enforceability of open source licenses, it seems that actual enforcement is largely para-legal. According to O'Mahony's empirical study, most community projects seek licenses compliance through informal

²⁹⁸ See e.g. St Laurent (2004), Guadamuz (2004), Malcolm (2003) and Metzger and Jaeger (2002)

²⁹⁹ Legal doctrines on binding contract and copyright license may differ significantly. See e.g. Rosen (2004). On arguments why contracts are better see Malcolm (2003)

³⁰⁰ See e.g. Moglen (2001a) and Guadamuz (2004), p. 334

means such as pressuring online discussions, emails and negative publicity. They try not to even threat with legal action or claim damages. Justification for enforcement comes from the collective prevailing opinion of the community.³⁰¹

One must be careful not to overemphasize the role of para-legal enforcement. While it may be quick and efficient and cover most license compliance issues among a close-knit community of developers, it still has a limited reach. If an accused license violator so wishes, he may finally seek for the opinion of the court of law. And then what matters is the legal reading of the licenses on the basis of intellectual property laws in force.

5.1.4 Licenses Categorized

Open Source Initiative has accepted tens of different software licenses as open source. While the number of approved licenses has been continuously growing, there haven't been much new inventions in the license functionality. Many of the latest licenses are project or company specific variants or combinations of already accepted licenses.

There are different ways to classify open source licenses. Metzger and Jaeger have used six different classes in their detailed legal analysis.³⁰² In this book, we use two classification systems: functional and historical. Based on functionality, we separate three different categories: licenses with (1) strong and (2) standard reciprocity obligations and (3) permissive licenses. Based on historical origin, we separate four categories. (1) GNU, (2) academic, (3) community and (4) corporate licenses. Let's define each category in turn.

Functional differences. From functional perspective open source licenses can be classified based on how each license treats the modification right of source code (derivative works). First, we define *standard and strong reciprocity obligation* as follows:³⁰³

³⁰¹ See O'Mahony (2003) who calls the informal mechanism as "the court of public opinion"

³⁰² See Metzger and Jaeger (2002).

³⁰³ Term "reciprocity" is derived from Rosen (2004). In economic literature, for example Leiner and Tirole (2005) use terms restrictive (standard) and highly restrictive (strong) licenses. It is

- *Standard reciprocity obligation* means that the distribution terms of the source code must be maintained. Such licenses are commonly called as *copyleft*. If the source code is developed further, the licenses terms can't be changed or the source code closed. However, if the source code is combined with another source code to create a new work, then standard reciprocity obligation does not apply to the combined work.
- *Strong reciprocity obligation* extends the standard reciprocity obligation: even adaptations and derivative works must keep the license terms intact. Such licenses may be described – depending on the value judgment – either strong copyleft or causing a “viral effect”³⁰⁴

Some licenses with strong reciprocity obligation have been extended further with a *network use obligation*. It is usually thought that licenses with standard and strong reciprocity obligations become applicable after the software is distributed and traditionally a distribution is interpreted as a downloadable or fixed software package. Network obligation adds that also the plain use of such software over a network should be interpreted as distribution. Hence, reciprocal licenses with an additional network use obligation can be described even “stronger” than licenses with the strong reciprocity obligation.

In addition to reciprocal licenses, another major functional category is *permissive* licenses. These allow free distribution, copying and modifying. Even change of license terms of adaptations of the original source code is allowed and therefore there are no reciprocal requirements in permissive licenses. Permissive licenses have no network usage clause.

To sum up the above, we can illustrate the functional differences between open source licenses in the following figure:

also rather common – also in academic literature – to call reciprocal licenses as simply “copyleft” licenses following the Free Software Foundation's policy objectives

³⁰⁴ On the one hand, open source critics may compare the situation to viral infection meaning that even if a new work is only partially based on source code under strong reciprocity it may not change the license terms, with standard reciprocity that would be allowed. On the other hand, Free Software Foundation uses term copyleft, which arguably signals their ideological approach to copyright.

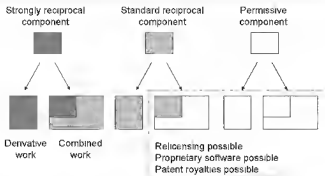


Figure 12 Functional differences regarding combination and modification between open source licenses.

A software component with a strong reciprocity obligation will maintain its license terms even when used as a part of a combined work (such as embedded software). Under standard reciprocity obligation, combined software can be re-licensed, i.e. the licensee is not tied to license terms. However, if the component itself is further developed, all derivative works must remain under the same license. Components under permissive licenses do not have any of these licensing restrictions. They can be combined in proprietary software and further developed under any conditions. – Network use obligation is not illustrated in this picture since it addresses the distribution and not combination or modification of the software.

Different historical origins. Many open source licenses carry a history load. From this perspective we can identify four major license categories: GNU, academic, community and corporate licenses. The differences here do not deal with the license functionality but are merely related to the readability of the licenses and the extent each license takes into account other rights and obligations in addition to the copyright to source code.

These more practical factors may be decisive in the license choice as we show later in this chapter.

The first category includes the so-called *GNU licenses*. Introduced by Richard Stallman and Free Software Foundation in the 1980s, these licenses carry a strong ideological message. The language of GNU licenses is written as to any "like minded" software developer and the licenses are already quite familiar among developers. Not surprisingly, as open source has become more common in larger organizations, lawyers have become hesitant to GNU licenses for their vague language and uncertain implications. Still, GNU licenses are being used for diverse goals as community-based Linux and corporate-oriented MySQL show as extreme examples. Unfortunately GNU licenses have also incompatibility problems with many other open source licenses – sometimes based solely on Stallman's uncompromising ideology. As would be expected, the licenses go explicitly against software patents: those bind by GNU licenses should license to anyone for free any patents that apply to GNU licensed software.

In the second category we have *academic or research licenses*. These licenses originate from large-scale publicly funded Internet infrastructure projects at US universities starting from UC Berkeley. For example major Internet-related components such as the domain naming system (BIND), internet-protocol (TCP/IP) and email processing software (Sendmail) have become *de facto* standards under permissive academic licenses. Academic licenses are short and rather clear in language although they do not take into account that many rights and obligations either that corporate counsels might find comfortable to live with. However, one could think an academic license is more often read than e.g. a GNU license and probably also understood by others than developers. Academic licenses do not typically take any stand on software patents or other intellectual property rights than copyright. They are mostly compatible with other open source licenses.

Third category consists of *community licenses*, which typically originate from some major free software project. They have gained popularity along with the Internet and free Unix implementations. Most popular is Artistic License originally distributed with Perl programming language. It is very

hacker-oriented with ambiguous terminology and many open interpretation issues. While it may fit perfectly the hacker culture, no corporate lawyer would perhaps ever suggest using such risky and uncertain terminology. Another popular is Apache license originally from the web server and related tools by Apache Software Foundation. Apache license is legally considerably more rigid than Artistic License taking also into account patents and trademarks. Finally, one could also include public domain into the community category though it can also form a separate group of its own.

When open source software became more popular in large IT enterprises in the late 1990s, many of them started to introduce licenses of their own. First major *corporate license* was introduced by Netscape in 1998 when they opened the source code of their popular web browser. Other large IT companies followed including IBM (Common Public License), Apple (Apple Public Source License) and SUN (Sun Public License and Sun Industry Standards Source License). Also Open Source Initiative itself introduced more corporate-minded open source licenses written in lawyer language (Open Software License and Academic Free License). Corporate licenses are typically very detailed addressing issues such as patent and trademark licensing, copyright to code contributions and many formal issues not included in other open source licenses. There are differences though; for example Sleepycat License likens other "pure" open source licenses in its writing style and has historical origins going beyond the Open Source Initiative.

5.1.5 Popularity of Open Source Licenses

Table 8 below features the most popular licenses at Sourceforge.net (as of late 2004), which hosts tens of thousands of free and open source projects:

License	Functionality	Origin	Popularity
GNU GPL	Strong reciprocity	GNU	66.5 %
GNU LGPL	Standard reciprocity	GNU	10.6 %
BSD	Permissive	Academic	6.9 %
Public domain	Permissive	Community	2.7 %
Artistic	Permissive	Community	2.0 %
Apache	Permissive	Community	1.9 %
MIT	Permissive	Academic	1.7 %
Mozilla	Standard reciprocity	Corporate	1.5 %
Common Public License	Strong reciprocity	Corporate	0.6 %
Zlib	Permissive	Community	0.5 %
QPL	Strong reciprocity	Corporate	0.4 %
Open Software License	Strong reciprocity	Community	0.4 %
Python License	Permissive	Community	0.4 %
Academic Free License	Permissive	Community	0.3 %

Table 8. Most used open source licenses on projects hosted at Sourceforge.³⁰⁵

Of all projects at Sourceforge that disclose license information, an additional 1.7 % has a proprietary license. These percentages only reflect the popularity of the licenses among new and non-mature projects. Many large and mature open source projects are coordinated through dedicated development forums. Also, license popularity only indicates their relative importance. Despite GNU licenses seem to be overwhelmingly popular, many important and central open source projects use different licenses as was noted above.

There are some empirical studies on who exactly use which kind of licenses. Lerner and Tirole found, based on an analysis of Sourceforge, that open source projects targeted towards developers and those projects, which produce basic Internet infrastructure software have usually permissive licenses. To compare, Sourceforge projects aimed at end-users usually have more restrictive licenses with reciprocity obligations.³⁰⁶ Further, Fershtman and Gandal observed interestingly that Sourceforge projects with permissive licenses generate more output per developer than projects with reciprocal licenses.³⁰⁷

³⁰⁵ In the table, particular licenses include all versions of it.

³⁰⁶ See Lerner and Tirole (2005).

³⁰⁷ See Fershtman and Gandal (2004). One reason can be that projects with reciprocal licenses have plenty of authors who only want their names listed. Instead, projects with permissive li-

5.1.6 A Framework for License Analysis

In what follows, some of the most relevant open source licenses in each category are introduced and analyzed from legal perspective. As the licenses change over time, the analysis does not aim at comprehensive and pedantic legal interpretation of the whole license texts. Rather, the aim is to identify how the main functionality has been implemented in the license text and then interpreted in practice. With each license the following things will be discussed in detail:

1. *Derivative works*. what kind of reciprocity obligation does the license have, if any?
2. *Patents*: how does the license treat software patents?
3. *Compatibility*. is the license compatible with other licenses?

Further, we discuss the issue of *intellectual property warranty* (indemnification) with those licenses, which have or have had such feature. We are also generally interested in how particular licenses have *evolved*. It can be assumed that some of the license features are more settled while others may need revision because of e.g. interpretation or compatibility issues. The general approach of the licenses towards *moral rights* is also shortly mentioned.

Before the analysis, it must be once again stressed that most open source licenses have not been tested in the court. While the licenses may well be enforceable according to the letter of law, court disputes on open source licenses have been extremely rare.³⁰⁸ Comparing different licensing terms from open source to proprietary ones, journalist Andrew Leonard has noted that "licenses are only as good as the faith that people put into them."³⁰⁹ Therefore, in the following, we refer in addition to license text

censes attract mainly those who really want to participate and learn from the development without opportunistic goals much like in ideal science

³⁰⁸ Some of the rare cases include *Progress Software Corp. v. MySQL AB* (2002) and a German court decision by *Landgericht München I* on 19th May 2004. In both cases GNU GPL was treated as valid.

³⁰⁹ See Leonard (2000b), comparing the ethics of open source licenses to those used by the entertainment industry.

and intellectual property laws to informal community norms, online discussions, and out-of-the-court disputes, which describe how the licenses are applied and interpreted in the real life.

5.2 GNU GPL and Strong Reciprocity

5.2.1 Derivative Works in Copyright Law

Many open source licenses include obligations to source code recycling based on the concept of derivative works in copyright law. But what exactly constitutes a derivative work of a computer program in the US and EU copyright law? The interpretation is not straightforward. First of all, "derivative works" is a strictly US legal concept. 17 USC 106 (2) defines it as:

"derivative work" is a work *based upon* one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications, which, as a whole, represent an original work of authorship, is a "derivative work".

In the United States case law, courts have determined that to be derivative, a computer program must be substantially *similar*³¹⁰ and in some form *include* a portion of the copyrighted work.³¹¹ To compare, in European laws there is no exactly corresponding wording to the US derivative works. Consider article 2 of the directive on the legal protection of computer programs, which states as a prohibited act:

³¹⁰ Litchfield v. Spielberg (1984). See also Lemley et al. (2000), p. 208-209.

³¹¹ It was held in *United States v. Manzi* (1995) that 70% similarity in the code base was sufficient to make the other work derivative. See also Lemley et al. (2000), p. 208, referring to case *Vault Corp. v. Quaid Software Ltd.* (1988) where the quality of the copying is discussed.

"(b) the translation, adaptation, arrangement and *any other alteration* of a computer program and the reproduction of the results thereof, without prejudice to the rights of the person who alters the program"

Interpreting the wordings, one could say that the European definition of derivative works is much broader and stricter: *any alteration* of existing works creates a "derivative work".³¹² Instead, in the United States, the new work must be *based on* the underlying work. In the context of computer programs, taking only a short passage of someone's copyrighted source code into a combined program could be considered as an "alteration of the work" while the new program might not be considered as being "based on" the recycled code.

Any copyright license must be interpreted in each jurisdiction under the applicable laws in force. As discussed, the US and EU copyright law differ in their respective formal definition of derivative works. Many open source licenses have been written with explicit reference to the US style wording but we may argue that the same wording covers also European definition although the concept of derivative works may have wider application in the EU than in the US.

Source code interpretation. Suppose now that both the European and US systems recognize the existence of derivative works. So how far does the control of the original author reach? Let's start assuming there are two source codes and it is claimed that one source code is a derivative work of the other. Lawyers have developed two theories applicable with the source code analysis of derivative works: the *idea-expression* dichotomy and the *abstraction-filtration-comparison* method.

The idea-expression dichotomy says that copyright applies only to expressions and not to ideas. Therefore any idea behind the work, say a mathematical algorithm or an idea to develop a new kind of word process-

³¹² In addition, it can be argued that the author's moral right of reputation extends the scope of the modifying right in Europe. See Metzger (2004). One can also state that modification does not threaten the reputation of the creator of an *industrial product* as much as the creator of a *fine art*. See Dietz (1994), p. 184.

ing software is not under copyright. In other words, copyright does not limit any subsequent author from developing a new operating system or compression algorithm. Existing programs can be studied and analyzed for the basis of new original works and only the literal copying of source or object code is restricted.

Abstraction-filtration-comparison method has been developed in the US case *Computer Associates International v. Altai*.³¹³ Ravicher has found evidence this method to be nowadays the dominant way of interpreting derivative works of computer programs in the US district courts.³¹⁴ The idea here is in short that the similarity between two source codes must be made by first abstracting the structure and functions in the suspected source codes, then filtrating inessential parts (non-copyrighted, public domain etc.) out and finally comparing the result. Comparison is not based on idea-expression dichotomy but to more detailed contextual analysis of source code structure, variable names etc. The method is illustrated in the figure 13 below.



Figure 13. Abstraction-filtration-comparison method.

Abstraction essentially results in a filtered version of the alleged derivative, which is then compared to the original. The case *CA v. Altai* relied heavily on the expert witness of computer science professor Randall Davis whose view was that a computer program consists of both text and behav-

³¹³ *Computer Associates International v. Altai* (1992)

³¹⁴ Ravicher (2002).

ior.³¹⁵ According to this view, the source and object code are copyrightable text while the actual operation of the program is more like behavior. Therefore for example interfaces and other behavioral functionality could not be regarded as text nor are they copyrightable.

Component based interpretation. Let's analyze the problem further in the terms of *component based* view of computer programs. In practice, the component based view may be more usable in open source development, which favors programming paradigms that rely on maximal source code reuse. Perhaps the most common problems occur when third party open source components are used. Figure 14 below illustrates the issue:

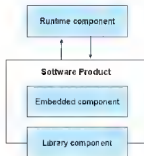


Figure 14 A simplified component-based view of a computer program.

Assume a developer has a software product where he uses third party source code distributed in a component. The question is if the resulting work combining the main program and the component is considered as a derivative work of the component and hence under partial control of the third party copyright owner. Let's separate the question further:

³¹⁵ Later Davis published an article on the issue coauthored with law professors. See Samuelson et al (1994).

- The source codes of the main program and the component have been mixed (an internal or embedded component). This kind of internal component tailored "inside" the whole work creates most clearly a derivative work of the combined whole.
- The product uses component's functionality (e.g. adds a new runtime-interface on the top of it). Since copyright does not cover use or interfaces, such external unmodified runtime components operating only through interface specifications would clearly not be derivatives of the product.
- Component has been linked to or linked from (e.g. an external library or device driver). This is the most vague case. Consider a software product that links to library routines. Does it become a derivative of the library? Maybe it doesn't; many library routines cover just standard functionality to help exactly in the "routine" tasks. But what if these routines cover essential and original functionality of the software and no other routines could be used with it? Further, consider a component according to a client-server setting where the component acts as a server to the client software product (component is linked from). Perhaps the software product would not be a derivative of the server component if it only uses the server's services. But if the product depends heavily on the server's functionality and does not run separately in any other setup without it, the situation is more dubious: if one wants to use the product, then also the component must be copied and distributed.

Communications based interpretation. The two interpretation criteria above are both based on the conventional wisdom of the contents of copyright law. While they may satisfy lawyers and project managers, more technical people may still continue to ask what really constitutes a derivative program between two seemingly separate program components. Free Software Foundation has suggested an interpretation based on more technical criteria and the mechanisms and semantics of communication between the components:

"What constitutes combining two parts into one program? [...] We believe that a proper criterion depends both on the mechanism of communication (exec, pipes, rpc, function calls within a shared address space, etc.) and the semantics of the communication (what kinds of information are interchanged) [...] So when they are used for communication, the modules normally are separate programs. But if the semantics of the communication are intimate enough, exchanging complex internal data structures, that too could be a basis to consider the two parts as combined into a larger program."

This interpretation suggestion can be illustrated in the following figure:

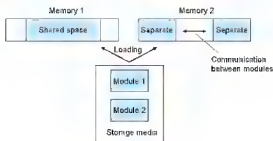


Figure 15. Derivative works and loadable modules

The question is whether modules are derivative of one another when they are loaded from storage media to the computer's memory. Basically in the case of shared address space the answer would be yes, and in the case of separate address space the answer would depend on the communications. Interestingly, Free Software Foundation takes a viewpoint of software as speech and communication. If one part of the program speaks with the other part with "physical contact", there could be a derivative work in place. From the perspective of copyright law, this interpretation is, however, questionable since copyright does not cover functionality but literal code.

It is not easy to compare the different criteria to analyze when two parts should be interpreted as creating one whole. Each of the criteria discussed here may apply better than others in different situations. There are also some clear limitations to the above criteria. In networked programming one can think of situations, where third party components have been mixed with other parts creating a "derivative work" with any of the criteria mentioned. However, the whole work is not under derivative work concept since it is only run and accessed through the Internet but *not distributed or copied* as a software package. This is common when a program is based on e.g. web services or other online components.

5.2.2 Derivative Works and GPL

Background. Perhaps the most relevant functional feature of GPL is the way it controls the further distribution of derivative works. In short, GPL requires that no one can change the license terms of derivative or modified works – otherwise redistribution is not allowed. The connection between GPL and derivative works is stated in the term 2b) of the license:

"You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications ... provided that ... :

2 b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License."

This part of GNU GPL has been used for numerous political debates. As open source was breaking through to mainstream in the software industry, most notably Microsoft argued that because of GPL 2b) any software dis-

tributed with it would seriously harm the software ecosystem. In 2001, Microsoft CEO Steve Ballmer famously claimed:³¹⁶

"The way the license is written, if you use any open-source software, you have to make the rest of your software open source... [GPL'd software] is a *cancer* that attaches itself in an intellectual property sense to everything it touches" (italics added)

Since then, Microsoft has somewhat mitigated their position. Open source supporters, and the software industry now more generally, argue that the scope of the reciprocity obligation in GPL is limited. In fact, most reciprocal license authors provide detailed guides on how to avoid a situation where the license would pose problems to the licensing of a derivative work.³¹⁷ So far, there is no evidence that open source licensors would use these obligations with malicious intention trying to turn all software into open source.³¹⁸

So, let's look at the term 2 b) in more detail: what does it exactly say? The wording includes both "derived from" and "in whole or in part contains" – the latter giving impression that GPL might cover more than the derivative work concept in copyright law implies. We can rephrase the question as follows:

1. When does a computer program *derive from* a GPL program? This is a quite straightforward link to the wording of US copyright law.
2. When does a computer program *in whole or in part contain* a GPL program? One may argue that this latter part creates a vague contractual definition of what GPL means by a derivative work: another program needs only to contain parts of source code under GPL to become governed by it. However, because there is no indication of the quantity or quality of the "contained" code and its relationship to

³¹⁶ The statement is commented e.g. in Greene (2001)

³¹⁷ For example GPL FAQ in FSF (2004) has currently over twenty detailed questions and answers regarding the interpretation of 2b)

³¹⁸ See e.g. Boyle (2004), arguing that the intent of GPL authors and judicial common sense speak against a scenario where any major software product would be required to use GPL if it would include some notorious amount of GPL source code

the combined work, it could be further argued that the interpretation of derivative works applies also to the second part of the question. It can hardly mean situations where non-copyrighted source code is used. After all, if we take GPL as a copyright license, it can only govern something that is copyrighted, i.e. original work of art.³¹⁹

Finally, consider the first part 2 b). It further limits the applicability of the license to those derivative works that are “published” or “distributed”. These both refer to well-founded legal concepts in copyright law. In the context of computer programs, however, their interpretation may not be that straightforward. For example, it is unclear whether selling a software subscription service means that the software is distributed or published at all.

Next, we discuss different practical interpretation situations where it has been not clear whether the whole work is a derivative of the component or any external source code under GPL. Some of the main issues have been:

1. Program output
2. Programming libraries
3. Plug-ins and device drivers
4. Client programs and user interfaces
5. Software subscription and web services

Program output. Consider a theoretical model of computer program as a black box taking input, making a computation, and producing output. One could say any output from a computer program is mechanically computed from the input. Therefore, it is worth asking if the output could be interpreted as a derivative work of input and computation.

The question is quite topical regarding program compilers. If compiler output is regarded as a derivative work of both the source code and the

³¹⁹ Further, the first term of GPL gives support to this interpretation defining that “a ‘work based on the Program’ means *either the Program or any derivative work under copyright law* that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ‘modification’” (*italics added*).

"work" of a compiler, could the author of the compiler have also rights to the resulting output? If the output does not include any code or expression from the compiler, the answer seems to be straightforward no. But what if the resulting program includes any expression added by the compiler? Then the question comes back to source code based interpretation.

GPL section 0 has a quite open-ended wording on the issue:

"...the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does."

Gnu Compiler Collection (GCC) happens to be in the category where code may be copied into the binary. Therefore it has a special exemption to GPL:

"As a special exception, if you link this library with other files, some of which are compiled with GCC, to produce an executable, this library does not by itself cause the resulting executable to be covered by the GNU General Public License."

While the issue of derivative works is compiler specific it is practically essential. No new programs can be made without a compiler and GCC works in a way that some common code will be copied to every binary compiled. Without this exception, it would be possible to build only GPL programs with GCC under a strict source code interpretation of derivative works.

Libraries. One frequently debated issue is if function calls to external library functions should be interpreted to make the whole program as a derivative work. Here, we can distinguish between two types of situations:

- Run-time linking when e.g. an operating system function call is used outside the executable

- Static linking when a function call is compiled into one executable and called within the program

Some time ago there was controversy over these issues and many community people claimed that run-time linking would not constitute a derivative work. However, a common understanding among free software developers seems to be now that both types of linking constitute the combined program to be a derivative work.³²⁰ For instance Metzger and Jaeger argue that if a function under GPL is *loaded* into the computer's memory *at the same time* with the main program and linked there to practically become a single executable then the whole work should be interpreted as a derivate of its parts.³²¹

On the other hand, there should remain areas where linking is allowed. Otherwise it would not be possible to build any application to a modern operating system without the explicit acceptance of the operating system copyright owner. GPL has solved the question in clause 3 as follows:

"As a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable."

There has been one court case where the linking issue was almost taken under closer analysis.³²² Progress Software Corporation had linked its Gemini table handler to MySQL AB's database under GPL. Technically speaking, the linking of Gemini could have been called static since it was compiled inside the MySQL binary distribution. Progress didn't however release Gemini's source code and the parties ended up in a court. The court stated in a preliminary injunction:

³²⁰ This is explained in detail e.g. in Free Software Foundation (2003).

³²¹ Metzger and Jaeger (2002), p. 44-45.

³²² See Progress Software Corp. v. MySQL AB (2002). The case has been commented by e.g. Magnus (2003).

"MySQL has not demonstrated a substantial likelihood of success on the merits or irreparable harm. Affidavits submitted by the parties' experts raise a factual dispute concerning whether the Gemini program is a derivative or an independent and separate work under GPL, [paragraph] 2. After hearing, MySQL seems to have the better argument here, but the matter is one of fair dispute. Moreover, I am not persuaded that the release of the Gemini source code in July 2001 didn't cure the breach."

The case was settled afterwards. The court was supportive to MySQL's argumentation that the linking method used by Progress indeed caused Gemini to be a derivative of MySQL. It is interesting to note, however, that it might be possible not to release the source code of a program under GPL without causing commercial damages. Also, any potential damages can be stopped when the source code is released on a later date (after requests from the copyright holder).

Plug-ins and device drivers. Consider next a situation where one develops a plug-in or device driver to a program under GPL. Is such a plug-in a derivative work of the main program and, hence, under GPL? Free Software Foundations Frequently Asked Questions list answers with an interpretation criteria based on substance and form. It states:

"If the program dynamically links plug-ins, and they make function calls to each other and share data structures, we believe they form a single program, so plug-ins must be treated as extensions to the main program. This means they must be released under the GPL

...³²³

There has been also relevant practical debate on the issue regarding proprietary kernel modules such as device drivers in Linux. Linus Torvalds, the main author of Linux kernel, has added the following statement to Linux's GPL:

³²³ See Free Software Foundation (2003)

"NOTE! This copyright does *not* cover user programs that use kernel services by normal system calls - this is merely considered normal use of the kernel, and does *not* fall under the heading of "derived work"."

This note has lead many to believe that device drivers, and plug-ins in general, are not derivative works. However, the note has little if any legal effect. Derivative works are ultimately defined in law and not in the license. Moreover, GPL license does not allow modification of the license and, even more importantly, Linus Torvalds himself is only one of the numerous copyright holders to Linux kernel – some of which are unknown. Referring to *estoppel* doctrine, some commentator think this note could be taken only as an indication that Torvalds is not going to support legal action against any proprietary Linux device driver manufacturers.³²⁴

Torvalds has somewhat clarified his position during the years. In 1995, he explained that kernel modules are "logically independent" from the kernel itself and that they can be seen as "use" rather than "linking" to the kernel. He thought that any device driver written for example to Unix could be ported to Linux without the need to use GPL.³²⁵ In 2003, Torvalds further explained that:³²⁶

"- anything that was written with Linux in mind (whether it then *also* works on other operating systems or not) is clearly partially a derived work.

- anything that has knowledge of and plays with fundamental internal Linux behaviour is clearly a derived work. If you need to muck around with core code, you're derived, no question about it."

Obviously, Linux should be today familiar to most developers and Torvalds' interpretation would hence mean that basically all new kernel modules in Linux should be under GPL.

³²⁴ Raymond (2001).

³²⁵ Message by Linus Torvalds to gnu.msc.discuss, 17th December 1995.

³²⁶ Message by Linus Torvalds to kernel discuss, 3rd December 2003

Client programs and graphical user interfaces. Assume one develops a commercial client program for another program under GPL. A practical example is a graphical client to manage MySQL databases. Is a client a derivative work and under GPL? MySQL AB's licensing page stresses the concept of distribution saying that "if you distribute a proprietary application in any way" then GPL becomes binding.³²⁷ Obviously, the company regards all clients as derivative works and in order to even use a client with other terms than GPL the developer of the client would need to buy a proprietary license from MySQL AB. In effect, they seem to interpret GPL as if a separate (based on both component based and source code interpretation) client program from their database server – used for commercial purposes – would be under the derivative works doctrine because of GPL 2b).³²⁸

However, neither copyright law nor Open Source Definition or GNU GPL license for that matter place any restrictions on pure software use. Thus merely using a work under GPL unmodified can't be the basis for a derivative work. So clients are free from reciprocity obligation? To this, MySQL AB answers – in line with Linus Torvalds – that if one needs their database in order to run the client, then one is basically also distributing MySQL database and GPL becomes binding. Other interpretation would be against the intention of GPL. A critic can still ask: what if the server software is unmodified? In the end, the question is then reduced to whether a client can be interpreted to be a derivative of the server in the sense of copyright law.

Software subscription and web services. Some commentators have noted that there is an "ASP loophole" in GPL. Based on our functional definitions above, GPL has only the property of strong reciprocity obligation and not that of network use. In essence, GPL applies only when works are distributed further. By using or running the program does not oblige

³²⁷ MySQL (2004a)

³²⁸ There is a commercial rationale for their interpretation: MySQL AB uses a dual licensing business model. They sell proprietary licenses to those users – typically someone embedding the database into another product – who do not want to be bind to GPL 2b). See section 7.2 for more detailed analysis

one to the terms of the license. It is also possible to modify the program and run it for private purposes without publishing the source code. This wasn't an issue when the latest version of GPL was introduced in 1991.

However, today unpublished software can be used commercially. In network environment one does not need to "publish" the computer program in order to use it. Most web server software runs practically hidden from end users accessing them on browsers. As Tim O'Reilly contests:

"...all of the killer apps of the Internet era: Amazon, Google, and Maps.yahoo.com. They run on Linux or FreeBSD, but they're not apps in the way that people have traditionally thought of... one of the fundamental premises of open source is that the licenses are all conditioned on the act of software distribution, and once you're no longer distributing an application, none of the licenses mean squat."³²⁹

This sort of private modification of GPL seems to be allowed under 2 b). There are no obligations to publish the software, which is required in order for GPL to be effective. But what exactly is software publication? Outsourcing or hiring programmers to develop software in-house most likely does not constitute a distribution. Instead, if one hires an independent developer or a company who may desire to license the software to other parties later, then GPL most likely requires the software to be further distributed to anyone with minimal copying costs. Still, there remains a possibility that no third party becomes aware that the software is developed and licensed under GPL.

Obviously, the next version of GPL may address this usage model. At the moment for example Affero Public License has included a network use clause in effect requiring web service users to publish modifications.³³⁰ – We return to the problem of network use obligation with Open Software License below.

³²⁹ McMillan (2003).

³³⁰ Affero's modified GPL is also supported by FSF (2002)

5.2.3 Patents and GPL

GPL license has a somewhat negative approach to patents. Its term 7 reads as follows:

"7. If, as a consequence [...] of patent infringement [...] conditions are imposed on you [...] that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program. [...]"

The preamble of GPL explains the motivation behind this obligation:

"[...] any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all."

In short, GPL has a built-in termination mechanism that does not allow the development of software that requires any kind of license payments for third party patents. In more technical wording, GPL is incompatible with patent licensing fees: if there is a patent for some software invention and that patent is not licensed for free to every GPL user forever, it is not possible to develop free software for that invention.

The termination mechanism has a limited reach. In theory, a patent holder who takes GPL software into use and then starts charging patent license fees from other users essentially terminates the licenses of all others

but itself. Further, a patent holder can naturally license the patent to the users of other software, who utilize the patented invention.

5.2.4 GPL and License Compatibility

According to Free Software Foundation, GPL-compatibility means that

"...you can combine code released under the other license with code released under the GNU GPL in one larger program... The GPL permits such a combination provided it is released under the GNU GPL. The other license is compatible with the GPL if it permits this too."³³¹

There are good reasons to use a GPL-compatible license. First of all, source code under incompatible licenses can't be reused in projects licensed with GPL. Second, since many developers prefer GPL, it may be that an incompatible license does not attract the highest possible number of willing contributors. There are many cases where large projects have changed their licensing policy to GPL-compatible. For example Mozilla, popular Python programming language and Apache projects have adopted new licensing schemes to be compatible with GPL.³³²

Incompatibility is also a problem to the developers of non-GPL open source software. In addition to proprietary licenses, GPL is in fact incompatible with many other popular open source licenses.³³³ The developers who don't want to switch from incompatible open source licenses can't benefit directly or indirectly (based on derivative works interpretation) from any GPL source code. This is why for example MySQL database, which is licensed under GPL, had to add the following exception to their

³³¹ See FSF (2003). To be precise, GPL section 6 explains explicitly "You may not impose any further restrictions on the recipients' exercise of the rights granted herein."

³³² See Wheeler (2004) for these and other examples. Some projects have simply dual licensed their source code with GPL in addition to an incompatible license while others have revised their licenses to be compatible.

³³³ See FSF (2004b) for a list. Almost all other reciprocal licenses are incompatible. Also some permissive licenses are incompatible because of e.g. additional patent or attribution requirements.

GPL license in order to allow PHP application development on top MySQL without GPL requirements.³³⁴

"As a special exception to the terms and conditions of version 2.0 of the GPL:

You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a "FLOSS Work") licensed under one or more of the licenses listed below in section 1, as long as:

1. You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,

2. all identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves, are distributed subject to one of the FLOSS licenses listed below, and ... the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections, and"

Basically, combination of GPL code under this exception with other open source licenses is possible – provided that derivative works remain under GPL. This kind of exception makes GPL to function as a standard reciprocal license towards incompatible open source licenses.

³³⁴ Shankland (2004a). PHP has its own license, which is open source but incompatible with GPL

³³⁵ MySQL (2004)

5.2.5 Other Licenses with Strong Reciprocity

Other important licenses with strong reciprocity obligations are for example Common Public License and Open Software License.

Common Public License. What makes Common Public License important is not its absolute popularity but the fact that it originates from IBM and became the first open source license Microsoft took into use.³³⁶

From functional perspective, Common Public License closely reminds GPL. Term 3 b) iv) of the license says that source code must be available:

"A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that ... its license agreement ... states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange."

The license further defines in the first term a contribution as constituting also derivative works:

"...changes and/or additions to the Program ...Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program."

A literal reading of the license indeed suggests a *strong reciprocity interpretation* similar to GPL³³⁷. However, the issue doesn't seem to be very clear at the moment. IBM's FAQ indicates that they may have had in mind something in the lines of standard reciprocity:

³³⁶ Lawson (2004) Rosen (2004) discusses CPL in length in his book stating (p. 163) that "Some amateurs believe they can write open source licenses. They should first read a good license like the CPL and ask themselves if they can do as well".

³³⁷ It should be noted that CPL does not have explicitly anything on larger or combined works.

"15. When I incorporate a portion of a Program licensed under the CPL into my own proprietary product distributed in object code form, can I use a single license for the full product, in other words, covering the portion of the Program plus my own code?

- Yes. The object code for the product may be distributed under a single license as long as it references the CPL portion and complies, for that portion, with the terms of the CPL"³³⁸

On patents, CPL is somewhat more aggressive than GPL. The first license grants all licensees a royalty-free patent license in term 2 and then in term 7 further states:

"If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed."

The *patent termination* triggers at the event of litigation and does not require that any patents would be valid. Further, the provision seems to cover even patents not related to the software. Thus, a licensor has to essentially choose between using CPL'd software and using patents (for licensing fees or suing for infringement).³³⁹ A GPL licensor, by contrast, can charge licensing fees and initiate patent infringement lawsuits against other users without losing itself the right to use the software. CPL thus goes further than GPL in this respect. It is possible that GPL will be developed in the future to include similar anti-patent functionality.³⁴⁰

Open Software License. There have been many initiatives to clarify the most used open source licenses. Perhaps most notably, Open Source Initia-

³³⁸ IBM (2002).

³³⁹ Rosen (2004), p. 172

³⁴⁰ Currently FSF's license commentary notes that while the clause makes CPL incompatible with GPL, "We don't think those patent license requirements are inherently a bad idea". See FSF (2004)

tive's legal counsel Lawrence E. Rosen has been developing Open Software License in more *rigorous legal language* to address especially the issue of derivative works and the strong reciprocity obligation in GPL. He argues for legal formality, constructing the license to be rather an enforceable contract than a simple copyright license.³⁴¹

"The OSL is intended to serve the same functions as the GPL except that it is a contract, and to be interpreted under contract law, rather than a copyright license."

In addition to formal legal clarity, OSL has been a platform to develop open source license functionality between the clash of community and corporate demands. OSL's take on derivative works goes as follows. In the first section, the license grants every user a right to prepare "derivative works" but only on condition that all such derivatives "shall be distributed under the Open Software License". Further, OSL also tries to extend its reach, through contractual means, into *network use*. In section 5, the license defines "external deployment" as a situation where the work is:

"... made available as an application intended for use over a computer network As an express condition for the grants of license hereunder, You agree that any External Deployment by You of a Derivative Work shall be deemed a distribution and shall be licensed to all under the terms of this License, as prescribed in section 1(c) herein." (*italics added*)

So what does the network use mean? So far, there are only a few remarks what it could mean. Rosen himself has stated that for example if an Internet search engine were based on a modified software component under network use obligation, it would not be obliged to release the source code because it is merely making information available, not software. However, when the search engine would allow private searches of users'

³⁴¹ Lawrence Rosen's message to license-discuss@opensource.org 28.7.2002

own internal pages then the clause would perhaps apply.³⁴² Whatever the correct interpretation, it seems obvious that such network use functionality goes further than what GPL and CPL require. It makes OSL more restrictive than typical strong reciprocity obligations. It can be also questioned whether network use provision is in accordance with the Open Source Definition, which requires there are no obligations for software use for example in commercial setting.³⁴³

Regarding patents, OSL includes a *simple free patent license* to all OSL users for those licensor's patents that cover the software. In earlier versions of the license (pre 2.1), there was a stricter termination clause, which terminated the license even when the licensor initiated a suit based on an unrelated software patent against open source users as with CPL term 7. Lawrence Rosen went on to change the patent license terminology according to the needs of the software industry (as software users):³⁴⁴

"...[patent termination] concern was most strongly expressed in an email from ... HP. I have since discussed this privately with attorneys for several other companies. I agree with them that a change is needed to make these licenses friendlier to companies that own large patent portfolios."

Maybe the most troublesome legal issue with open source licensing is *intellectual property infringement liability*. The core of the problem is potential strict liability for copyright and patent infringements that affect every party taking into part in the software distribution. The basis of the problem is in the logic of intellectual property laws. It does not help the distributor that he has been in *bona fide*, good faith. If the work infringes a third party intellectual property right, everyone that participates in the distribution chain of the work may be liable to the infringed author.

³⁴² See Rosen's message to license-discuss@opensource.org 5.11.2004. To be precise, Rosen describes the latter example as a situation where the user would upload the material to the servers of the search engine developer.

³⁴³ Section 6 of OSD reads "The license must not restrict anyone from making use of the program in a specific field of endeavor" and clarifies that this means especially commercial use. GPL states that "The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program"

³⁴⁴ Rosen's message to license-discuss@opensource.org 25.3.2004

Most open source licenses disclaim all warranties including potential intellectual property infringements. From version 1.1 onwards, Open Software License however *includes a limited IPR warranty* for third party copyright and patent claims. This is unheard of in other open source licenses. In term 7, it states:

“Licensor warrants that the copyright in and to the Original Work and the patent rights granted herein by Licensor are owned by the Licensor or are sublicensed to You under the terms of this License with the permission of the contributor(s) of those copyrights and patent rights.”

Rosen has argued that the licensor is in a better position to judge whether there are infringing contributions and recommends all projects using OSL to make sure contributions have no IPR infringement risks.³⁴⁵ Such a rationale can be easily opposed: why should an open source developer warrant anything for which he does not take price in the first place? Why should an open source developer take an additional intellectual property infringement risk regarding for example software patents he is not aware of and can't control how the software will be used? – We return to this issue below with Creative Commons licenses and explain why that particular licensing project decided to drop a somewhat similar warranty provision. In chapter 6, we continue the intellectual property infringement discussion in more detail.

5.3 GNU LGPL and Standard Reciprocity

5.3.1 LGPL Functionality

GNU LGPL (short for Lesser General Public License) differs from GPL in functionality. First, it has only the *standard reciprocity obligation*. This means that direct modifications to LGPL software itself must be redistributed under LGPL (or GPL) but combinations of LGPL software with other software can be distributed even with proprietary licensing terms. The li-

³⁴⁵ Rosen (2002).

cense is aimed specially for programming libraries. This is explained in term 6:

"...you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice".

It seems that linking is possible without other conditions than that users must be always able to modify the library itself. Therefore, the source code of a LGPLed library must be separately available with instructions on how to re-link the library with the main program.³⁴⁶ In practice, the architecture of a closed source main program may require re-designed to allow re-linking with LGPL source code separately from the main program.

Otherwise, LGPL compares to GPL in functionality. Patent clause is similar and the language of the license in general is mostly exactly copied from GPL. Thus, there is no need to discuss additional features.

Licenses with standard reciprocity obligations are generally speaking *more compatible than strong reciprocal* licenses although they are not free from problems. If LGPL code is linked into a larger work, the license terms of that larger work have no obligations from LGPL and the licenses are compatible. However, when an external component is linked into a LGPL'ed work then LGPL is obviously incompatible with the other license in the case the other license does not allow LGPL to override. In practice, then LGPL seems to be as incompatible as GPL in the case the reciprocity provision applies.

Also, LGPL has evolved to better meet the criteria of its developer, the Free Software Foundation. The first version of LGPL from 1991 was short for Library General Public License and recommended for all programming libraries. A new version came out in 1999 when Free Software Foundation started to discourage its general use because of its limited reciprocity. Richard Stallman explained the motivation, which lead to the license name change to Lesser General Public License:³⁴⁷

³⁴⁶ LGPL term 6 has numerous details on how this should be realized

³⁴⁷ Stallman (1999a)

"Which license is best for a given library is a matter of strategy, and it depends on the details of the situation... we are now seeking more libraries to release under the ordinary GPL... Using the ordinary GPL for a library gives free software developers an advantage over proprietary developers: a library that they can use, while proprietary developers cannot use it... when a library provides a significant unique capability ... releasing it under the GPL and limiting its use to free programs gives our community a real boost... University projects can easily be influenced; nowadays, as companies begin to consider making software free, even some commercial projects can be influenced in this way."

While Stallman's intentions are understandable, they may discourage the use of GNU licenses in the software industry at large. Taking GNU licenses into use may mean that in the future the license terms are changed to reflect the goals of Free Software Foundation, which may not be in par with the industry understanding of intellectual property and software business. No wonder many corporate users have written their own company specific reciprocal licenses.

5.3.2 Other Licenses with Standard Reciprocity

GPL with library exception. Since linking with LGPL may be problematic in practice, FSF has an exception template to be added to GPL, which clearly allows all kinds of linking situations (both dynamic and static) without any obligations. It was first issued with Guile project. The exception is now used more generalized for example in the GNU Crypto project as follows:

"As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license

of that module. An independent module is a module, which is not derived from or based on this library. If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version."

In short, linking libraries under GPL with such a library exception does not oblige one to stay with GPL. The functionality of this kind of amended GPL is similar to that of LGPL without some of the interpretation problems. It seems license variability is not always a good idea especially if licenses become too complex.

Mozilla Public License. Mozilla's license was originally crafted in 1998 to govern the distribution of Netscape's open sourced Internet browser. One could describe it as the first corporate open source license. MPL has a *standard reciprocity* provision in term 3.2:

"Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License"

The requirement does not reach to derivative works and does not constitute a strong reciprocity obligation. Term 3.7 reads:

"You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code."

Larger Work is further defined as:

““Larger Work” means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.”

MPL has an explicit patent license, where contributors agree to grant users unlimited licenses for the patents they owns that apply to the whole source code. MPL has also a *patent defense clause* in term 8 is, which is again more extensive than the one in GPL. It aims to create a defensive copyright portfolio out of MPL part of the software. If a third party patent holder initiates court action against the program authors, then all rights potentially granted to the patent owner regarding the software will be terminated as with Common Public License. If the patent holder wants to reconsider the situation, the term gives him 60 days to either withdraw the suit or pay licensing fees.

MPL has again severe incompatibility problems. To be sure, the incompatibility only applies when source code is literally mixed and the reciprocity provision applies. In that case the license is for example incompatible with all the licenses we have analyzed so far: GPL, OSL, CPL and LGPL. The incompatibility was a reason for re-licensing the whole Mozilla source code with a multiple license scheme in 2001:³⁴⁸

“It is unclear whether a developer could be successfully sued for copyright infringement on grounds related to these perceived license incompatibilities. However, to eliminate possible uncertainties concerning this question, and to address the concerns of developers who wished to use Mozilla code in applications whose code was otherwise licensed under the GPL or LGPL, we decided to seek relicensing of the Mozilla code to address the perceived license incompatibilities for both the GPL and the LGPL.”

Such a decision has not been trivial to implement since the project had already a number of outside contributors. After three years, the relicensing

³⁴⁸ Mozilla Relicensing FAQ (2004)

is still ongoing.³⁴⁹ For its part, the Mozilla relicensing example tells about the costs of license incompatibilities, how important it is to control rights ownership and make sustainable licensing decisions.

5.4 BSD and Permissive Licenses

5.4.1 BSD Functionality

Among different open source licenses, BSD is the oldest and most well known permissive license. Originating from university environment, BSD can be described to reflect the principles of *academic freedom*. The license allows, but does not require, the source code to be open. Redistribution of software under BSD can be distributed also in binary form without source code.

BSD must not be confused with public domain. There are two minor requirements for binary distributions with other licenses. First, the BSD license including the names of the copyright holders and warranty disclaimer must follow with the distribution.³⁵⁰ Second, the license explicitly requires that the author names can't be used in endorsing any program derived from the BSD source code. It's doubtful, though, whether such endorsement would be possible even without such explicit prohibition.

In other words, BSD license guarantees attribution to all contributors but protects them from liability and bad reputation following the moral rights principles of copyright. These requirements apply also to derivative works whether they are made through linking a module under BSD into a larger work or developing the source code under BSD further.

It must be noted that BSD does not guarantee that the source code stays open or that there will be new additional terms in a direct copy or a totally new license for a derivative or combined work. Thus, any BSD licensed component can be basically re-licensed in a new combined or derivative work, which may carry any other license from reciprocal open source to

³⁴⁹ See Markham (2004) calculating that around 2% of the Mozilla code is still left to re-license and then calling for help noting that "those [parts of source code] left are nasty or irregular in some way."

³⁵⁰ In fact, all the other open source licenses discussed above also include somewhat similar attribution clauses.

proprietary licenses. This also means that BSD license is *compatible with almost any other license*.

Positive endorsement requirements of permissive licenses have, however, implied incompatibility problems in the past. The original BSD license from 1989 included a clause requiring anyone using that software to expressly mention University of California in the software documentation or associated materials. This little detail made Free Software Foundation to declare BSD license incompatible with GPL. In effect, the advertisement requirement was dropped in 1999.³⁵¹

BSD license is noticeably *silent on patents*. Commentators have argued that the license includes an implicit patent license: developers must give necessary patent licenses to all software users since otherwise the use would not be possible.³⁵² Perhaps there is something in the fact that the license over fifteen years old.

5.4.2 Other Permissive Licenses

MIT License. MIT Public License is another very popular permissive license and has almost the same structure as BSD license though it is even shorter. The main difference to BSD license is that MIT license lacks the non-endorsement requirement. Since there are no other differences in functionality to BSD, there is no need to discuss MIT license details further.

Apache License. One of the most elaborated permissive open source licenses is the latest version 2.0 of Apache license. Earlier versions (1.0 and 1.1) of Apache license were literally very similar to BSD but the new version is substantially more detailed. However, functionally Apache license does not still differ drastically from BSD. Changes in the license language reflect increasing corporate concerns in open source licensing.

Regarding derivative works, it is possible to use other licenses for Apache code. Different from BSD and earlier Apache licenses, this option is stated explicitly in the last paragraph of term 4:

³⁵¹ FSF (2003).

³⁵² E.g. Rosen (2004), pp 78-79

"You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License."

The license has an attribution requirement similar to BSD. It has also a non-endorsement requirement through term 7, which states that the license does not include any implicit or other right to use trademarks and product names of the software.

Again different from BSD, Apache license has a *detailed patent clause*. First, any contributor must license his patents applicable to the software without any royalty requirements. Further, the patent license has a far-reaching termination clause much like CPL and MPL.²⁸³

"If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed."

The license has a standard warranty disclaimer. As a new feature, Apache license states explicitly the option to provide additional warranties in term 9:

"While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License."

²⁸³ See Apache (2004) for a discussion on whether this clause is compatible with GNU GPL.

Artistic License. Artistic License was introduced by Larry Wall in March 1991 as part of Perl 4.0. Reason was that GPL didn't suit all corporate needs since it was not possible to use Perl for proprietary programs. To maintain full compatibility with GPL projects, Perl has been since dual licensed both under GPL and Artistic License. Larry Wall has commented his intentions as follows:

"The intent of the Artistic License has never been to be watertight. I'm happy if the Artistic License conveys my intent to honest folks, and at the same time gives corporate lawyers the warm feeling that they could wriggle out of it if they really wanted to"³⁰⁴

As Wall's note suggests, Artistic license might be more hacker oriented and tricky to understand than others. Artistic license is basically a permissive license but with multiple options: user can also choose to be bind by strong reciprocity obligation if they wish so.

Commentators have criticized artistic license because of its lacks in legal form and vague language. There have also been attempts to clarify the license although the proposals have not become that popular. However, the license is a good example of the still existing role of developers in writing licenses for central and important open source projects.

5.5 Excursion: Creative Commons Open Content Licenses

5.5.1 Background

During the recent years, the basic principles of open source licenses have been adapted also to other copyrighted works than computer programs. Traditionally, licenses to artistic works such as music and pictures (content, in short) have been managed by copyright collecting societies. However, on the Internet individual authors can also distribute their works directly to users. Also users' expectations to copy, modify and distribute works on the Internet may significantly differ from the physical world. This has opened an opportunity for new licensing initiatives.

³⁰⁴ Larry Wall's message to gnu misc discuss 6 4 1994

It is difficult to estimate the popularity of different open content licenses because of the short time they have been used. It seems clear, however, that the number of high quality open content projects is quickly increasing.³⁵⁵ The most popular licensing initiative so far has been Creative Commons (CC).³⁵⁶ CC project was started in 2001 as an initiative to standardize more liberal license terms in content. Formally, CC is a Massachusetts-based non-profit corporation with a few employees working at Stanford University in California and an uncountable number of volunteers contributing to the project over the Internet. Major United States universities have since started to advocate CC with Stanford University's law professor Lawrence Lessig in the highlight.³⁵⁷ First versions of the licenses were released in December 2002 and new updated 2.0 versions in May 2004. An increasing number of websites and content on the Internet use CC licenses.³⁵⁸

This section discusses briefly the main features CC licenses and the organization of the CC project. There are many similarities but also some clear contrasts to open source. The following discussion aims to point out some of the limits when the principles of open source are generalized outside the realm of the software industry

5.5.2 Creative Commons Functionality

In practice CC works as an Internet service for the creation of copyright licenses in content. Users make a few choices and can then view suitable licenses. Licenses have three representations: (1) technical rights description, (2) detailed legal license text, and (3) short explanation of what the li-

³⁵⁵ One of the most prominent projects is Wikipedia, an online encyclopedia based on user contributions, which started in 2001. Wikipedia aims to make proprietary encyclopedias such as Britannica obsolete within the next 5 years. See "Wikipedia Founder Jimmy Wales Responds", Slashdot, 28th July 2004. Available at <http://www.slashdot.org/>

³⁵⁶ See <http://www.creativecommons.org/>

³⁵⁷ Lessig is well known by his popularized books on law and technology. His latest work addresses especially problems with media ownership of the culture and introduces CC as one proposal for a change. See Lessig (2004).

³⁵⁸ In 2003 Creative Commons licensed one million works. By May 2004, the total number of CC-licensed works was over three millions. A well over 50% of all CC-licensed works are text, the rest being roughly equally divided to sound, images, video and "interactive". See Mike Linksvayer's message on 11th June on Creative Commons discussion list available at <http://lists.biblio.org/pipermail/cc-licenses/2004-June/000953.html>

cense means. There are a number of different CC licenses available. Published works are then linked to the selected license located at CC website.

License terms. All CC-licenses have similar structure including terms *common* to all licenses and selected *specific* terms. In common terms, all CC-licenses allow copying, distribution and public performance and display of the work without any license payments. In other words, the licenses give users rights without obligations. Works under CC may be used, copied and distributed further without additional permissions or license fees. In addition, CC common terms state that the licenses do not interfere with fair use rights (such as citations, private use etc), first sale or the freedom of expression. Moreover, common terms state that works can not be used with digital rights management systems, which may limit any right granted in the licenses including fair use, first sale and the freedom of expression.

In addition to the common terms, any CC-license may have one or more of the following interchangeable specific terms restricting the use of such works to some extent:





-  **Attribution.** You let others copy, distribute, display, and perform your copyrighted work - and derivative works based upon it - but only if they give you credit.
-  **Noncommercial.** You let others copy, distribute, display, and perform your work - and derivative works based upon it - but for noncommercial purposes only.
-  **No Derivative Works.** You let others copy, distribute, display, and perform only verbatim copies of your work, not derivative works based upon it.
-  **Share Alike.** You allow others to distribute derivative works only under a license identical to the license that governs your work.

Figure 16. Possible terms with related logos in CC-licenses.

Additionally, there are several specific CC-licenses the number of which is supposed to grow. In this short excursion we can only briefly discuss some of the most interesting ones.³⁵⁹ "Public domain" and "founder's copyright" address a shorter expiration for copyright public domain would expire immediately and founder's copyright after 14 years.³⁶⁰ Also the most popular open source software licenses GNU GPL and LGPL are available from Creative Commons as "CC-GPL" and "CC-LGPL" branches. Technically, these external licenses are linked from Free Software Foundation's website and CC only adds to the work a CC-symbols, summary of the license and technical rights description.

Comparison to open source. While CC and open content in general has much in common with open source, there are certain differences. For instance, software authors themselves have written many popular open source licenses. They have codified the existing sharing culture of computer programmers and, thus, open source licenses have not needed much enforcement. Instead, CC has taken a strict *top-down approach*. The licenses were carefully prepared and marketed by an entity specifically founded for that purpose.³⁶¹ This may affect license interpretation: there do not, as of yet, exist such community norms as with open source licenses.³⁶² It is also interesting to note that most CC-licenses go explicitly against the Open Source Definition in restricting for example the commercial use of works.³⁶³

³⁵⁹ Licenses that can be only mentioned include "Re-combo" or "Sampling", which clarifies the legal status of creative modification of musical works, and "Share music", which basically allows music to be non-commercially copied and distributed. In addition, a recent large-scale initiative "Science Commons" aims to answer to the open access debate bringing CC principles to academic publishing. See e.g. Nature (2001) on "open access" discussion.

³⁶⁰ In the United States, the copyright term was initially 14 years when copyright was for the first time taken into use in 1790. "Founder's copyright" therefore refers to the term the "Founding Fathers" of the United States agreed upon. It should be noted that it is legally unclear whether one can withhold from copyright and release the work to the public domain before its expiration.

³⁶¹ To compare, in principle anyone can submit a new license to Open Source Initiative to be certified to comply with Open Source Definition. Creative Commons doesn't have such process.

³⁶² For example, most free software users accept Free Software Foundation's interpretation of GNU GPL license as stated in the FAQ on their website.

³⁶³ Open source definition, section 6, does not allow discrimination against any type of use. This includes discrimination against commercial use of the programs.

5.5.3 Risk Allocation and Warranties

We noted above that Open Software License differed from other open source licenses in that it offered an intellectual property infringement warranty. Also the first versions of CC-licenses include a clause shifting the burden of third party intellectual property claims to the original licensor. They state:

“By offering the Work for public release under this License, Licensor represents and warrants that ... 1. Licensor has secured all rights in the Work necessary to grant the license rights .. without You having any obligation to pay any royalties ... 2. The Work does not infringe the copyright, trademark, publicity rights, common law rights or any other right of any third party or constitute defamation, invasion of privacy or other tortious injury to any third party.”

Beneficiaries of this kind of warranty clause are for example different middlemen and service providers who can take licensed works into use and distribute them further with lower risk. If third party rights have been violated, the author is in the end liable for the infringement. Unfortunately, such a warranty clause is far from bullet proof. If the author is unknown or bankrupt, the burden of third party liability will be practically on all those authors and users who are sued. This can be quite unjust especially for users and authors acting in good faith. Under free licensing systems, they don't ask license fees for copies but they may still be held liable for copyright infringements.

The intellectual property warranty was removed from 2.0 versions after heavy criticism by licensors.³⁶⁴ Nevertheless, the basis of the problem is in the liability rules of copyright law, not in the licenses. Unforeseeable liability remains as one of the things that may stifle the development of large-

³⁶⁴ According to the license revision announcement the warranty provision was removed because “Ultimately we were swayed by a two key factors (1) Our peers, most notably, Karl Lenz, Dan Bricklin, and MIT (2) The realization that licensors could sell warranties to risk averse, high-exposure licensees interested in the due diligence paper trail, thereby creating nice CC business model.” See Brown (2004)

scale open content and open source projects alike. The risk of liability for violating third party intellectual property increases as the source code or content is scattered. This makes it hard to start open projects that involve many participants and potential right holders such as open movies. – We continue the liability discussion in the next chapter.

5.5.4 Internationalization and Formalities

Adaptation approach. Creative Commons is the first major open licensing initiative, including both open content and open source, which aims at license internationalization. The CC leaders who come mainly from the US academic legal community believe the license texts must be translated to national languages and adapted to national jurisdictions. An assumption for internationalization is that an English language license based on US copyright law text may not be valid in other countries.

One problem with national adaptations is possible inconsistencies. Indeed, a quick comparison shows that there seems to be substantial differences in practice.³⁶⁵ Obviously, the CC project gave substantial freedom to each national internationalization team. Many, but not all, explicitly state that the license should be interpreted as a contract.³⁶⁶ Some translations include notorious terminological changes; for example instead of distribution they may speak of making publicly available. In many cases, the definitions are taken from national copyright laws. In most adaptations, fair use is edited to match the European copyright laws, which typically include a closed long list of limitations to the exclusive rights.³⁶⁷ An explicit reference to database right is added on some, but not all adaptations. Only few licenses take the issue of moral rights explicitly into consideration.³⁶⁸

Forum shopping. Internationalization through translation and legal adaptation makes the licenses more understandable and also legally valid in more jurisdictions. However, the approach has also clear drawbacks. As a

³⁶⁵ See International Commons at <http://creativecommons.org/projects/international/>

³⁶⁶ At least Spain, Holland and France

³⁶⁷ British version includes an appropriate reference to "fair dealing".

³⁶⁸ Interestingly Australian version is taking moral rights most explicitly on the table

practical matter, the usability and interoperability of licenses may suffer because users have to deal with a number of different license versions in different languages with different terminology. As a legal matter, the CC-licenses state:

"You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons Commons license that contains the same License Elements as this License"

Consider that one goes to CC-website and chooses to license the work with a CC-license adapted and translated into Finnish. According to the cited term above, that does not mean that in Germany the work would be used with the Finnish version of the license. Thus, the author can't control which version of the licenses the user is going to choose. In the end, users may have possibilities to "forum shopping" inside an internationalized open content licensing project – as in the real world

Comparison to open source. It can be argued, however, that license internationalization through translation and legal adaptation may not be crucial for the actual success of open licensing. The most popular open source software license GNU GPL has been used since 1989 all over the world without any known legal case, where the license or a part of it would have been judged invalid.³⁶⁹ The fact is that also CC licenses are being applied in all jurisdictions. At least so far, the actual behaviour of users has counted more than detailed legal interpretation and risk assessment.

³⁶⁹ It must be admitted that there is not much counterevidence either, in one recent case GNU GPL was held specifically valid in a German lower court decision by Landgericht München I on 19th May 2004. See the decision at http://www.jbb.de/urteil_lg_muenchen_gpl.pdf

5.5.5 Concluding Remarks

The time has proved the early prognoses of copyrights death on the Internet exaggerated.³⁷⁰ Instead of collapsing, copyright law has been complemented by more liberal licensing practices on the Internet. In a way, the copyright law has been fixed on its own merits.³⁷¹ However, there remain many kinds of legal challenges on the way the problem of strict liability in copyright law being perhaps the most troublesome. In the long term, if Creative Commons and other open content licensing models get increasing support from the surrounding media industry, there should be room for a change in the law. – We return to this issue at the end of the next chapter.

There is also the challenge of attitudes, community creation and the necessary technical infrastructure. Leading a huge project on open content licensing requires community support. Creative Commons top-down leadership model might be its weak point that may alienate some authors. In addition, it is still rare to attach rights descriptions to works published on the Internet and most users don't read complex license texts. Only education and time will tell if the open content licensing challenges will be really adopted by the masses the way open source licenses have been adopted by software developers.

5.6 Summary: Competition Between Evolving Licensing Standards

New open source licenses and rewritten versions of old ones obviously will come up. One could argue that in the long run *open source licensing has become more corporate-friendly* and many licenses have evolved to answer the needs of the software industry. The licenses now generally address the existence of software patents in detail. The limits of derivative works have been tailored to new software uses on the market. In side with the commercialization trend, also the ideologically loaded GNU licenses have sur-

³⁷⁰ For example Negroponce (1995), p. 58, famously claimed that "Copyright law is totally out of date. It is a Gutenberg artifact. Since it is a reactive process, it will probably have to break down completely before it is corrected"

³⁷¹ Lessig (2004) has noted that CC tries to complement rather replace the current copyright system. See also Merges (2004), noting that the claimed expansion of intellectual property rights may have been balanced to some extent by the recent open licensing initiatives.

vived and proved they have a role in the markets. GPL is no longer referred to as a cancer of the industry.

In side with the evident evolution in licensing details and techniques, however, the main functionality of the licenses as abstracted in this section has not changed that fast. Each license can be with relative ease classified into certain well-defined group. The following table summarizes from the foregoing presentation the functional features of some of the most popular licenses from copying, distribution and modification (derivative works) perspective:

	Free distribution	Free use	Open code	Standard reciprocity	Strong reciprocity	Network reciprocity
Proprietary	-	-	-	-	-	-
Shareware	X	-	-	-	-	-
Freeware	X	X	-	-	-	-
BSD, MIT, Apache	X	X	X	-	-	-
LGPL, MPL, ...	X	X	X	X	-	-
GPL, CPL, ...	X	X	X	X	X	-
AfferoPL, OSL, ...	X	X	X	X	X	X

Table 9 Copyright functionality in different license types

Regarding other relevant license features, we can conclude that:

- Only the latest industry prepared open source licenses have *explicit patent and trademark* licensing and termination requirements. It can be assumed that future licenses will continue to have such explicit terms.
- When intellectual warranty provisions have been proposed, they have soon been *removed* because of strong community disapproval
- All licenses include explicit attribution requirements, and some even extended reputation requirements, thus *enforcing the moral components* of copyright

When a new software project is initiated, *licenses do compete*. The competitive situation has both positive and negative implications. For one, licensing options are *good for the variety* of individual open source projects.

As we noted earlier, projects may have different ethical, technical and business goals, which can be satisfied most concrete at the level of license terms. However, competing situation combined with the fact that many open source licenses are in practice incompatible with each other may in the long term lead to open source projects, which are not able to benefit from development efforts in other projects. In short, this may have *negative effects to the open innovation model* that has been identified with the success of open source development. Developing Open Source Definition towards compatibility standard with the cooperation of license authors could be a long-term solution to the compatibility problems.³⁷²

³⁷² Open Source Initiative has considered amending Open Source Definition to stop license proliferation. See Russell Nelson's email to license-discuss@opensource.org March 2, 2005. This does, however, affect the compatibility issue only indirectly.

6 DEFENSE WITH OPEN SOURCE: INFRINGEMENT RISK MANAGEMENT AND PATENTS

Any use of open source in business environment includes legal risks. In this chapter we discuss how software patents and other intellectual property infringement risks can be managed at individual firm and social policy level. We start from the more general intellectual property rights infringement risk management alternatives in open source development. From there, the discussion is extended to the social problem of software patents especially in the European policy context.

6.1 How to Manage IPR Infringement Risks?

6.1.1 Background

Next we discuss defensive rights management alternatives for open source developers. The recently much debated legal cases initiated by SCO against large Linux users are perhaps the best examples of how intellectual property rights can be – in theory – used as a strategic tool against particular software users. Open source, and distributed components based software development methodology in general, highlights the risks of potential third party copyright and patent infringements.

While SCO case is not a good example of a successful IPR offensive because SCO has been unable to prove actual infringement, it has forced many companies to rethink their intellectual property defense strategies. It is very well possible that some companies have hidden and more easily verifiable rights to existing open source products, which may prove troublesome in the future. For example, a leaked memo from Hewlett Packard shows how seriously a large IT enterprise takes the potential risk of patent infringement lawsuits against open source.³⁷³

³⁷³ See Barr (2004), for a memo from July 2002, where HP executives assume that “Microsoft is about to launch legal action against the industry for shipping Open Source software that may force us out of using certain popular Open Source products”

Theoretical framework of this section is mainly based on the economic theories of product liability³⁷⁴, liability insurance³⁷⁵ and patent strategy³⁷⁶. Taking a practical approach, we try to build a forward-looking framework for the management of such risks in open source development. We also test the framework presenting evidence on actual risk management practices gathered from publicly available market news, interviews and patent statistics.

For the purpose of our risk management analysis, we identify three types of different organizations who develop open source: IT enterprises, open source companies, and community projects. With *IT enterprise* we refer to established big companies who have numerous product offerings in the computer and software markets including significant interest and involvement in open source software. Typical IT enterprises are for example Apple, HP and IBM but also Microsoft because of their role as a kind of counterpart to open source. Second, with *open source company* we refer to any small or medium size company that develops and markets software products based on open source components. Pioneering in this sense are those companies, which are build around well-known open source products such as Red Hat Inc. (GNU/Linux) and MySQL AB (MySQL). Third, with *open source community project* we refer to non-commercial but significant open source development projects such as Linux or Apache.

This section is organized as follows. First we discuss the nature of potential third party copyright and patent infringements in open source software. We see them as accidents towards the users, which are caused by the actions of the right holder. Then, we four different defensive risk management alternatives are introduced: (1) liability allocation in software license, (2) liability insurance, (3) acquiring patents, and (4) risk avoidance. To test the relevance of the framework, actual risk management policies are reviewed. We find that the risk management strategies of established IT enterprises, pioneering open source companies and open source community projects each differ from the others. Obviously, an optimal strategy depends on the actual infringement risk, which may be significantly dif-

³⁷⁴ E.g. McKean (1970), Calabresi (1970), Eppler and Raviv (1978), Viscusi and Moore (1993).

³⁷⁵ E.g. Shavell (1982), Sarath (1991), Winter (1991)

³⁷⁶ E.g. Teece (2000), Shapiro (2001), Bessen (2004)

ferent for various types of developers. Finally, the chapter concludes with a discussion on the implications to long-term risk management and licensing policy.

6.1.2 Nature of Third Party IPR Infringements

Infringements as Accidents. Basic relationships between involved parties in an infringement situation are characterized in the following figure:



Figure 17. Developer-chain and IPR infringement.

Assume there is an open source product, where Developer₁ (acting in good faith) adds a contribution, which infringes IPR holder's copyright or patent. Now, every subsequent developers from Developer₂ to Developer_n, can be liable according to copyright and patent laws even if they did not know that the software infringes a third party right. Developers can violate copyright by e.g. copying, modifying and distributing the infringing source code contribution. Developers can also infringe patents by e.g. making, selling and commercially using patented inventions embodied in the infringing contribution. In essence, such direct intellectual property infringements are under a strict liability rule.

First, we believe that the situation of an individual developer towards users (any subsequent developer or the end-user) is significantly analogous to a product liability setting. Therefore, we start by analyzing *IPR infringement as an accident* resulting in a negative externality for all open source developers and their users. Our question is, following Calabresi's

seminal treatise on the economics of accident law, what are the costs of such accidents and how open source developers can minimize them within the developer-chain.³⁷⁷

Second, we open up the “black-box” of infringement claim and ask how the infringement risk can be managed towards the *cause of the accident*, namely the IPR holder. The question then becomes, what means an individual developer has to protect itself from unwanted actions from the side of IPR holders.

Before our analysis of risk management alternatives, a more detailed explanation of the nature of IPR infringements as accidents is in place. We aim to demonstrate that since the infringements are hardly avoidable *ex ante*, the accident analogy is indeed applicable.

Copyright. It is possible to think of two types of copyright infringements. Either source code has been copied in ways not permitted in the copyright law or license terms have not been honored.³⁷⁸ Infringements may be in both in-house or third party produced components. Problems with the latter are naturally more difficult to identify

Literal source code copying can be avoided to some extent in advance. However, it requires that the suspected copied source code is available, which typically is not the case. Also technical analysis of source code copying does not go very deep. There are clear limitations for example in identifying structural copying.

Also license term infringements can be avoided to some limited extent in advance. A prerequisite is that all licenses in a given product can be listed from external files, source code etc. Further, the known licenses can be evaluated for possible third party rights reservations and other non-standard terms. Unfortunately, the interpretation of license terms may not be clear especially if individual program authors have written them. As noted, there is also disagreement in the interpretation of many standard open source licenses. Not surprisingly, Free Software Foundation enforces

³⁷⁷ See Calabresi (1970)

³⁷⁸ In fact licenses can be also based on other rights than copyright but for the purposes of this section it is sufficient to address licenses simply as based on copyright

out-of-the-court over fifty GNU GPL license violations annually and estimates that they are becoming more commons.³⁷⁹

From the above, it follows that an open source software developer or user who utilizes third party components has *only very limited means to avoid copyright infringement risks in advance*. If the source code of the component is not fully available, it is practically impossible to evaluate the risk. The availability of the source helps infringement analysis theoretically according to the rough lines explained above but in any case the analysis will be costly, time-consuming and far from complete. Source code availability may also actually rise the infringement risk since it makes easier for offensive rights owners to identify infringements.

Patents. In addition to copyright, there may be hidden patents, which are not known until the software has become popular and well known. The risk of patent infringement grows with the geographical market area and is obviously greatest in the United States markets. It is *however difficult to estimate the real risk of patent infringement*. Obviously many commonly used open source software components already infringe patents. For example Open Source Risk Management Inc., who sells IPR insurances, claims that Linux kernel could infringe almost 300 different patents. – Earlier they declared Linux does not infringe copyrights.³⁸⁰

As noted, the identification and evaluation of patent infringement risk is in practice close to impossible. Since most patent claims do not include source code, it is not possible to do technical searches based on available patent data and open source code. In practice it may be possible just to avoid infringing well-known patents and follow the patent portfolios of close competitors. Although, even keeping up with the competitors may prove both prohibitively costly and difficult because of the dynamism of software industry.

Already thousands of software patents have been granted in different parts of the world. It would be unrealistic to claim that even the largest IT enterprises could know with precision whether their large products in-

³⁷⁹ Cohen (2003)

³⁸⁰ OSRM (2004a), OSRM (2004b)

fringe third party patents. However, large enterprises have by definition better means to defend themselves against possible infringement claims. They typically own patent portfolios that can be used strategically for counterclaims and cross-licensing. Little open source companies, which lack extensive patent portfolios, need other defense options.

In short, *even careful developers and users regardless of their size have more limited possibilities to avoid patent than copyright infringements.* The risk is essentially bigger for the developers of products with large market share. Paradoxically, these are also the developers and programs that are perhaps socially most beneficial. Again, open source code does not help in defense but merely increases the infringement risk since patent owners may more easily identify infringements.

6.1.3 Alternatives to Manage Risks

License Disclaimers and Warranties. According to the cheapest cost avoider principle, the costs of accidents should be allocated to the party who can avoid them with comparatively lowest costs.³⁵¹ For example if we assume that developer has superior knowledge of the risks involved in his software, he should also bear the most risks from accidents resulting from its use. Having some, but not excessive, liability increases investments in new product development and also the overall product quality.³⁵² In software development, however, the standard liability rule has been *caveat emptor*, i.e. the subsequent developer or finally end-user is liable.

The end-user liability rule may not be economically efficient, if the user has substantial transaction costs in finding the actual level of infringement risk. In many cases such information is simply not available. On a positive note, user liability may increase competition if users are more cautious in finding the safest products and there is no product, which would be seen as inherently less risky. Moreover, in some cases users are better able to determine the actual case-by-case risks as they have better information on how they are going to use the software (e.g. internally, in public).³⁵³

³⁵¹ Calabresi (1970), McKean (1970).

³⁵² Viscusi and Moore (1993)

³⁵³ McKean (1970), Eppler and Raviv (1978)

Historically, liability limitations were needed in software licenses mainly because developers wanted to avoid the risks of any damage caused by technical errors or bugs. It has been usually argued that it is not practically possible to write complex bug-free programs.³⁸⁴ In addition, since each program copy is identical to the other, any error will multiply, and potential liability burden for developer would be huge. Requiring stricter liability would therefore imply significantly higher software prices without much increase in quality (no new functionality, only less bugs).

Also open source license disclaim any kind of warranty for programming faults or bugs. These risks are shifted down from the original developers to subsequent developers and end-users. Most popular open source licenses limit also the developer's liability from legal errors including copyright and patent infringements to the maximum extend permitted by the law.

Drawing a difference between technical and legal errors is rare in open source licenses. As we noted in the previous chapter, Open Software License includes a more specific infringement warranty (indemnification) clause that does shift more liability towards original developers. In practice, this means that if a subsequent open source developer intends to warrant itself from IPR risks against the original developer, an additional indemnification clause need not be negotiated. Such a clause is efficient only to the extent the original developer has resources to defend the user in court. In the event the original developer is unknown or bankrupt, the clause has no effect whatsoever. This leads us to discuss the third party insurance alternative.

Additional Insurance. Independent of the liability allocation, insurance is arguably a desirable alternative from social policy perspective.³⁸⁵ Ideally, when compensation maximum has been agreed on, that can be used further as a liability limitation in license and contractual obligations. If developers have the liability, they would carry only the risk of paying deducti-

³⁸⁴ E.g. infamously by Brooks (1975).

³⁸⁵ Shavell (1982).

ble in the event of a proven infringement. Users would participate in insurance payments by paying slightly higher prices for their software.

An extreme option would be a *public insurance system* much like the car insurance systems commonly used. However, that would also imply high administrative costs and obviously software development would have to be monitored by further government rules. Perhaps some development methods would be banned altogether.³⁶⁶

Compared to the liability limitation clause in a license, insurance surely costs more and may not be available to non-commercial community developers. In theory, insurance is economically rational choice when accidents are rare and difficult to estimate but when the event finally occurs, the results can be financially devastating. Insurance is also a mean to price IPR infringement risk more objectively on the markets.

Pricing such insurances is however problematic in practice. Since e.g. patent infringement cases are rare, there is little precedent to efficiently price IPR liability insurances. Another problem is that the developers have superior information about the risks, which is costly to acquire independently by the insurers. In such a new market, insurance providers also limit their responsibility and it may be impossible to get an insurance that would cover all types of IPR risks in all possible markets. This also seems to be case: for example patent litigation liability insurance has not been an option mainly because *the supply of such insurances has been low*. There is however some evidence, that the supply would be increasing.³⁶⁷

The economic theory suggests that if liability insurance is not available to all, then also litigation incentives should be cut.³⁶⁸ If incentives to litigate are by contradiction on the rise, then markets are inefficient and we must seek for the next risk management alternative. Enter patents race.

Strategic Positioning in Patents Race. Many economists have compared software patenting to an arms race where all market participants try to patent as much as they can. The result has been an extensive *patent thicket*, whose effects are further amplified by the sequential and cumulative na-

³⁶⁶ McKean (1970)

³⁶⁷ Betierle and Davison-Jenkins (2001), CJA Consultants (2003)

³⁶⁸ Sarath (1991)

ture of software development.³⁸⁹ The effects of patent thicket on innovation and licensing are not clear but it has been suggested that they can be in fact negative.³⁹⁰

In such an environment the developer has, besides extensive own patenting, basically three options: (1) acquire adequate patent portfolio by using mergers and acquisitions (2) purchase required amounts of patents from competitors, or (3) license the required patents either directly or from a patent pool.³⁹¹ Of course, the developer can also decide to do nothing and deal with the situation only if it ever emerges.

These options can be feasible for large IT enterprises, which often use either of them to capture and protect their position in different areas of software development. However, the first two options are typically very expensive and thus not even in theory available for typical open source developers. The last option is more common but not very viable for most of open source developers, which rely on GNU GPL as their license of choice. The reason is obvious: under GPL the licensee has to get a license for all future uses including derivative works. Patent pools do not normally offer that kind of licensing option, at least by default.

But how threatening are patents after all? It has been commonly argued that a mere infringement claim does not yet mean that open source could not be used. It has been estimated, that in the United States only 1,5% of all patents are ever litigated and as few as 0,1% are litigated until trial.³⁹² Further, as many as 46% of all patents litigated to trial are finally held invalid.³⁹³ When we also take into account that the most potential court case targets have patents of their own for defending purposes (cross-licensing and counter suits), then perhaps *ignorance of the whole patents race may be a reasonable strategy for smaller companies and community projects.*

³⁸⁹ See Bessen (2004) for an overview. The concept of patent thicket is not new. A similar development has been seen with many other emerging technologies such as semiconductors (Lemley 2002) and copying machines (Melamed and Stoepelwerth 2002).

³⁹⁰ Shapiro (2001), Bessen and Hunt (2003).

³⁹¹ See e.g. Bednarek and Ineschen (2003).

³⁹² Lanjouw and Schankermann (2001) and Lemley (2001).

³⁹³ Allison and Lemley (1998). This has led Lemley and Shapiro (2004) to characterize patents not as exclusive but rather as "probabilistic" property rights bearing similarities to lottery tickets.

Risk Avoidance. Finally, one may try to avoid IPR risks as much as possible. Within the development process, risk management policies can be improved by using formal written copyright assignments and prior patent searches and assurances from contributing developers. To be sure, an "avoidance at-all-cost" strategy can't be effective because, as we noted, many infringement risks simply can't be avoided in advance. Also, if open source products become standard, as has already happened in e.g. Internet technology, avoidance would be socially inefficient.

A long-term "environmental" avoidance strategy is to influence the legal policy to limit the reach of IPRs. Such *lobbying* is typically done through industry pressure groups or by helping grass-root activists. It makes economically sense to invest in lobbying as long as the marginal benefits (in this case lower IPR-risk) exceed the marginal costs.³⁹ There are some problems, though. One is that the political system is sometimes resilient towards lobbying and it is hard to predict the outcome. Also, in the case of IPRs there are typically strong opponents, which may neutralize any efforts.

Summary. Table 10 below summarizes the scope, effectiveness and costs of the discussed risk management alternatives.

Option	Scope	Effectivity	Price
Disclaimer	Licenseses	Low	Low
Insurance	Market	Relative	Relative
Patenting	Market	High	High
Defense alliance	Market	High	Relative
Avoidance	Market	Low	Low

Table 10 Comparison of different IPR defense options for open source developers

While a liability disclaimer in license is easy to add, it does not protect the developer towards IPR holder. Insurance is relatively more effective and can, at least in theory, protect developers against both licensees and IPR holders. However, in practice the IPR liability insurance markets are not very well formed yet. Patenting is seen as a highly effective risk

³⁹ Landes and Posner (2003)

management option though it costs undoubtedly more than any other alternative. Also, the real infringement risk of patents can be questioned. Finally, the rationality of avoidance should be judged against the lost benefits from using the risky software and lost resources in long-term political lobbying.

6.1.4 Actual Management Practices

In order to understand the actual management practices, qualitative information was collected from public press releases and interviews on the Internet and patent statistics. As noted, open source developers were divided into three categories: (1) large IT enterprises who sell "solutions" and large open source installations, (2) small pioneering open source companies who typically have an innovative open source product, and (3) community-lead open source projects without a company background. Next, we discuss the risk management practices of these three different types of organizations towards users (costs of accidents) and third parties (causes of accidents). Our aim is to preliminary test the feasibility of the theoretical framework.

IT Enterprises. When Utah-based software company SCO claimed in spring 2003 that Linux infringes their intellectual property, large IT enterprises selling and supporting Linux systems didn't respond immediately. Soon, however Hewlett-Packard started a new era by offering a limited warranty to its customers in fall 2003.³⁹⁵ Novell and Red Hat followed in the beginning of 2004.³⁹⁶ This way intellectual property warranties become a kind of *additional warranty business* for large IT enterprises.

Admittedly, these warranties are far from perfect: HP indemnifies only claims (both copyright and patents) in SCO case, Novell has limited indemnification to copyright claims, and Red Hat only promises to change infringing components without additional costs to the user. In addition, according to a recent survey, most corporate Linux users have not ex-

³⁹⁵ HP (2003).

³⁹⁶ Novell (2004), Red Hat (2004)

pressed interest in these programs.³⁰⁷ It seems evident that such warranty policies only reflect how risk-averse IT enterprises are towards IPR infringements.

Some IT-enterprises have joined Open Source Development Labs to create a joint Linux legal defense fund for Linux end users. The fund is not tied to any single vendor's Linux-distribution, but only supports end-users against SCO's suits, which confines its scope significantly.³⁰⁸

The patenting activity of some relevant companies as well as the approximate size of their patent portfolios is presented in the figure 18 and table 11 below:³⁰⁹

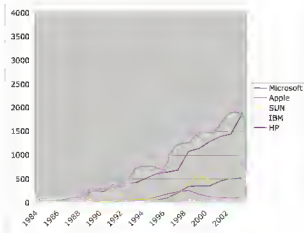


Figure 18 US patents granted between 1984-2004 Source: USPTO

³⁰⁷ Fuchera (2004).

³⁰⁸ Open Source Development Labs (2003).

³⁰⁹ Patents data was collected from US Patent and Trademark Office's online database. In the figure, HP includes both Hewlett-Packard and Compaq. Data includes all kind of patents granted to the named companies; most of the patents granted in recent years could be categorized as "software patents".

IBM	33665
HP	13696
SUN	4062
Microsoft	3359
Apple	1776
Novell	444
Red Hat	1

Table 11. Total number of US patents granted from 1984 to early August 2004. Source: USPTO.⁴²⁸

Known open source promoters IBM and HP have clearly the biggest patent portfolios though most of their software patents are from the late 1990s. Many early patents cover only hardware. It is worth noting that while the *patenting activity in general has increased* during the last few years, Apple Computer has been granted significantly fewer patents now than five years ago. It is also interesting to note that Novell has just fewer than 500 patents total and Red Hat, which should be counted as an open source company, was granted its first patent in 2004. Still, these two companies feel confident to offer IPR insurances for their Linux customers.

It seems evident that *patenting will increase in the future*. For example Microsoft has announced to apply for over 3000 patents, which means multiplying its current patenting activity a few times.⁴²⁹ It obviously aims to reach the level of IBM and HP.

However, there seems to be no immediate risk of patent infringement claims by the biggest patent owners. First, all of them subscribe to the *mutual defense policy*. IBM has said to use its patents only to defend itself in potential open source cases. For instance, after SCO had sued IBM for Linux infringements, IBM filed a countersuit in terms of three patent infringements.⁴³⁰ Also HP has committed to open source and has prepared to defend itself against possible patent offensives.⁴³¹ Even Microsoft has publicly said they look only for licensing possibilities, not attacks.⁴³² Second,

⁴²⁸ The number of those patents whose annual fees have been paid, and are an active part of the company portfolios, is obviously lower (Langouw and Pakes 1998).

⁴²⁹ Fried (2004).

⁴³⁰ Shankland (2004b).

⁴³¹ Barr (2004).

⁴³² Stone (2004). Of course, none of these statements has any legal effect.

some of the companies have started to *dedicate patents royalty-free* to the use of open source developers. In January 2005, IBM donated 500 patents to open source developers, then SUN granted 1200 patents with their Solaris announcement, and according to the latest news Computer Associates is considering to follow.⁴⁰⁵

Open Source Companies. Interestingly, many open source companies offer extensive warranties for their products. For copyright infringements, warranties sound justified, since source code rewriting is common.⁴⁰⁶ Also license infringement seems unlikely since these companies should have better knowledge of open source licensing. But for third party patents, infringement risk is harder to manage.

According to a recent EU commissioned study, patent infringement insurance has become a viable patent defense option to European small and medium size companies. The availability and terms of such insurances vary however greatly country by country and in practice they may not cover litigation in the United States, where the coverage would be most relevant. The patent infringement insurance market within the United States is even less developed.⁴⁰⁷

Red Hat was the first open source company to publicly announce it files for defensive patents in 2002.⁴⁰⁸ One lesson from the patenting arms race is that it takes years to build a credible patent portfolio by filing patents. The first patent was granted to Red Hat in 2004, four years after its actual filing date. Also, using the patents is relatively more costly to small companies as they are targets of patent lawsuits more often than enterprises having large patent portfolios.⁴⁰⁹ Not surprisingly, many open source companies oppose patents strongly at the policy level.⁴¹⁰

⁴⁰⁵ See IBM (2005), SUN (2005) and Computer Business Review (2005). To be precise, SUN's pledge is limited only to the users of their own (incompatible) open source license while IBM's pledge covers all open source licenses accepted by the Open Source Initiative.

⁴⁰⁶ Valimaki (2003b).

⁴⁰⁷ CJA Consultants (2003).

⁴⁰⁸ Red Hat (2004b).

⁴⁰⁹ Lanjouw and Schankermann (2004).

⁴¹⁰ E.g. Farber (2003).

Community Projects. The best option to community projects seems to be to keep liability disclaimers intact and avoid patented and proprietary technology with reasonable means. As a practical measure, for instance, the Linux development process was recently modified to better document the origin of the source code and any subsequent changes to it. The aim is to avoid legal uncertainties in advance. Now all Linux developers must click through a statement saying that to their best knowledge the new source code is home made, not copied from anyone, and that its distribution in Linux is allowed with GNU GPL.⁴¹¹ – It is worth noting that this “assurance” does not carry any kind of legal effect though.

Open source and free software activists have been circulating the idea of creating a patent pool for GPL-software for some time now. For example, Richard Stallman has been supportive for the idea, but so far Free Software Foundation has not made any formal steps to create a “GNU-patent pool”.⁴¹²

Open source community developers also *campaign visibly against stricter intellectual property legislation*. Many developers have warned that software patents and increasing proprietary control threaten the functionality of the open source model. – We continue from this in the next section.

6.1.5 Concluding Remarks

The use of extensive license disclaimers is still the standard among open source developers. However, we found that both IT enterprises and open source companies have started to sell liability warranties to their users. The growing warranty business proves that more research on risk management would be relevant to both developers and business managers.

The insurance alternative is still in large part an untouched ground. Open source companies are only slightly but perhaps increasingly interested in defensive insurance plans. The biggest challenge may however be non-commercial community projects, which are currently not able to gain directly from expensive insurances.

⁴¹¹ OSDL (2004).

⁴¹² Kelly (2000)

The defense options to patent infringements should be studied further. It is clear that IT enterprises have collected large patent portfolios consisting of typically thousands of patents during the last ten years. They have publicly announced to use them only defensively towards open source. Also open source companies are investigating defensive patenting possibility. While a mutual defense policy seems to be accepted by almost all companies, we did not find evidence of concrete IPR defense alliances in open source development. Perhaps patent pools have been an overlooked option. If pools would be constructed taking into board all parties, including non-commercial open source developers, they would surely lower the social costs of patent infringement risks.

Also the relevance of patents in open source needs further empirical study. For example the so-called intellectual capital management literature stresses the role of patents especially as a source of licensing revenue and gaining access to the innovations of others.⁴¹³ Instead, it seems more proper to analyze patents in open source development rather as risks than opportunities although we also noted that the costs of patent risks may not be as high as commonly assumed.

To summarize, in addition to liability allocation *inter partes*, third party insurance, and patenting, the risk management game in open source development includes careful risk avoidance and even political influence on the development of intellectual property laws. Like Calabresi noted in his analysis of accidents, it is plausible to assume that an optimal IPR risk management strategy for each type of organization is a mix of available alternatives.

6.2 Patenting Problem and Possible Policy Solutions⁴¹⁴

6.2.1 Background

One could argue that open source forms perhaps the fastest growing and most innovative sector of the software industry today. However, many individual software developers and small companies have warned

⁴¹³ See e.g. Teece (2000) and Glazer (1998).

⁴¹⁴ This part was originally published in Vålumäki (2004b).

that software patents – originally aimed at promoting innovation – threaten the functionality of the open source model.⁴¹⁵ Recent political campaigning around the EU's software patent directive proposal has increased fears that distributed open development is particularly vulnerable to patents.⁴¹⁶ Well-known open source developers including Linus Torvalds have expressed critical public opinions on software patents in general and the directive proposal in particular.⁴¹⁷ The critique has been mainly centered on economic arguments: extensive software patents may threaten software innovation.⁴¹⁸

Now we review several suggestions to solve the patenting problem with open source development. These include patent pools for open source developers, aligning development more closely with the patenting process, solving the problem of trivial patents and introducing new liability exceptions to the patent law.

We conclude by arguing that in the case patents and open source development have conflicts, the legal system should be improved in the first place. It would not be desirable to require open source development as a methodology to adopt abstract rules of law that may not be in par with the software development reality. Therefore the liability exception policy might offer the best outcome for all.

6.2.2 Open Source Licenses and Infringement Risk

Let's start from looking at the patenting problem from open source licensing perspective. As noted in the previous chapter, many open source licenses including GNU GPL, CPL, LGPL, MPL and Apache have a built-in

⁴¹⁵ Warnings have a long history in developer communities: there has been campaigning against software patentability since the early 1990s. See e.g. The League for Programming Freedom (1991).

⁴¹⁶ See Software Patent Directive Proposal (2002). Most active critique has been arranged by Foundation for a Free Information Infrastructure, a group run by Hartmut Pilch. See <http://www.ffii.org/>.

⁴¹⁷ For example Linux developers Linus Torvalds and Alan Cox sent an open letter to the European Parliament in the fall 2003 expressing their deep concerns. See Torvalds and Cox (2003).

⁴¹⁸ Political debate at EU level on software innovation is not new either. Lobbying on the Software Copyright Directive, centered on the questions of whether reverse engineering and copy right over interfaces would threaten software development and innovation. See Band and Katoh (1995) and section 4.1.3 above. What is new in the patenting debate, however, is the strong role of individual developers and activists.

termination mechanism that does not allow the development of software that requires any kind of license payments for third party patents. Of course, not all open source licenses have such patent clauses. For instance the popular BSD license lacks one. However, as we noted, patent termination clauses are becoming more and more common. Whatever one may think of the practicality of them, it seems clear that open source development becomes problematic indeed if there are many existing software patents around.

We have also noted that the biggest challenge with software patents is that the infringement risk is difficult to measure and manage *ex ante*. It is true that statistically patent infringement cases are rare, many patents are held invalid and that a mere infringement claim does not yet trigger the termination clause for instance in GNU GPL.⁴¹⁹ In practice, however, an open source project faced with a patent infringement claim from a credible corporation may have to terminate just because it would be too costly and time-consuming to find out what the real risk is. In addition to developers also open source users face the infringement risk. Open source licenses do not help since they typically spread the risk from any intellectual property rights infringement to the user.

So what are the practical options to react when a patent infringement claim arrives? First, the use of the invention can be stopped. Second, the patent can be analyzed and determined whether either a license should be negotiated or a new implementation around the patent written. While legally perhaps the safest option, writing a new implementation takes resources and some extra motivation might be needed to "invent the wheel" again. Licensing has its problems too: the patent license should be practically free-of-charge because of the terms of open source licenses (such as GPL clause 7 noted above) and because most individual developers and non-commercial projects couldn't afford any fees in the first place. Now how many patent-owners would be ready for that? Finally, it is difficult to argue why anyone should buy a license to an invention, which he cannot in many cases even utilize (there is no source code in patents).

⁴¹⁹ For example Rosen (2004) has stressed that because of these reasons open source developers should not be too worried about patents

6.2.3 Development Process from Patenting Perspective

From development perspective, open source and free software are ambiguous concepts. There are both commercial and non-commercial open source development projects. Some projects are coordinated by a legal entity, some by informal groups of developers.⁴²⁹

Common to all open source development is the transparency and incremental nature of all development. This means that:

- All source code – including potential software inventions – is public and commented from the beginning; there are *no periods of secrecy* and neither is patent data searched before publishing
- Development is *distributed* in a sense that the number of contributors is in principle unlimited and their identity may be anonymous
- Development is *incremental* meaning that typically contributions only cover a specific part or function of the program. A project may be abandoned by the initial developers and later continued by others.

There may be both individuals and organizations participating in the development process. The hierarchy and organization of the development process may not be visible to outside. A typical open source project does not require high resources to be technically efficient. Also, a successful project may gain high visibility and large user base without significant commercial marketing efforts.

From the short description above, we can find numerous reasons why the patent system does not fit open source development process at all. Most important problems have been said to be.

- Because of open source code patent infringements are relatively easy to detect and prove

⁴²⁹ A good overview of open source development process is presented in e.g. Feller and Fitzgerald (2002)

- Low resources do not allow patent search and legal defense in the event of infringement claims based on trivial patents
- Especially free software ethics and philosophy are strongly against the use of patents of any kind in software development
- Source publication on the Internet may be interpreted to happen in all jurisdictions and hence infringe potential patents anywhere
- Proprietary software developers may compete against open source developers with patent infringement claims; if their patents are held valid and open source developers would require to obtain licenses, there is no guarantee on the terms of these private license agreements between open source development community and commercial software company

It should be stressed that open source development has so far worked fine and produced new innovations without anyone applying or licensing patents. It is not in the scope of this book to study whether the patent system works efficiently with software development in general. It should be clear, however, that the potential setbacks to open source software development caused by an inefficient and malfunctioning patent system are far greater than they would be to other areas of innovative activity.

6.2.4 Policy Debate on Open Source and Patents

As noted, critical policy debate on software patents has been active for years.⁴²¹ The opponents of software patents – mainly individual open source developers, activists with various backgrounds, and small companies – have gained high visibility with comparably lower resource to patenting advocates. The debate is currently most relevant in the EU where several consultation and research reports have been published on the issue.⁴²² Suggested solutions in the recent studies and reports have varied from:

⁴²¹ See e.g. Nichols (1998), p. 103.

⁴²² See e.g. Bakels (2002), Blind et al (2001), PöT Consultants (2001), and Hart et al (2000).

- Solving the problem of trivial patents (Bakels)
- Founding a patent pool for the open source community (Pbt Consultants)
- Adjusting the development method to include patentable research (Nichols)
- Special liability exceptions to open source developers (Blind et al)

Let's discuss each of these proposals in turn from open source developers' perspective. First, solving the problem of trivial patents may not help at all. While much of open source code may be classified generic there are major projects, which produce new state of the art (operating systems, databases). Also proponents must quite optimistically assume that patent system works efficiently in software development and, if it does not, it can be easily fixed. However, as already noted in this study, it is unclear whether patents promote innovation in the software industry in the first place. Rather the patent system may benefit the marketing, financing, litigation and global business strategy of large companies.⁴²³

Any suggestions that open source developers should use the patent system have more fundamental problems. First, many open source licenses give patents no value: they essentially require all patents relating to open source software to be licensed to anyone free of charge. It is practically impossible to change such a well-laid principle in all licenses whose rationale may be strongly ethical.⁴²⁴ Second, though more theoretically, a large patent pool with restricted admission to "open-source-only" participants could be held violating the competition law. For a patent pool in open source world to be functional, it should accept as its members companies, individuals and even anonymous developers as well as both free and proprietary software developers.

Next, while adjusting software development to the model assumed by patent law may function in large companies it can hardly fit the informal and distributed nature of open source development model. It should be

⁴²³ See Chapter 3 and e.g. Bessen and Hunt (2003) and David et al (2003)

⁴²⁴ See also Stallman (1999b)

also noted that even large companies depend on external help with patents⁴²⁵

Finally, Blind et al. propose that in the future it should be considered whether non-commercial use of open source software should be exempted from patent claims – even if it takes place within commercial environment. This approach has its obvious advantages. First of all, it would be a commonly accepted social policy. Second, and perhaps more importantly, open source development as a methodology would not need to adapt to the abstract rules of law, which may be far from the software development reality. Instead, laws would be adapted according to an alternative development method.

6.2.5 Liability Exceptions for Open Source?

Let's think in practical terms for a moment. Individual developers and small companies have good arguments that the open source development model suffers from software patents. Their argument has its roots much deeper than the current public debate over the proposed directive. Their argument does not stop with the directive.

Since open source forms an essential and growing part of the software industry today, it is in the interest of all participants, big IT enterprises alike, to find a solution to the worries of patent opponents. Arguing that the patent system will work in the future and increase innovation in the software industry is not an answer to their problem. The question is about open source development and licensing model, which is in conflict with the software development and licensing model assumed by patents.

What realistic proposals do we have for those affected by patents? From social policy perspective, writing new exceptions in a new law seems like the best possible answer. Unfortunately the political debate in the EU has been going forward and backwards without clear answer to those worried about open source. Currently, the exception approach is at risk⁴²⁶

⁴²⁵ Blind et al (2001)

⁴²⁶ Initially, the commission proposal didn't include too many exceptions for open source. EU parliament changed the tone by voting for numerous exceptions to the directive proposal in the fall 2003. In spring 2004, Ireland's new "compromise proposal" revised most of these changes out. As of late 2004, the debate goes on

Finally, a quick speculation is in place. What would happen if the exception approach would win in the end? Quite interestingly, Blind et al go on to note that if a liability exception to open source is sought then also the TRIPS agreement would need similar revisions. Here, one may compare the position of open source advocates to those of developing countries. Both have general intellectual property (software code and indigenous cultural knowledge), which multinational corporations try to claim as their private property. The difference is that open source advocates work mainly in the developed world and their relative economic and political power may be far greater than those of the third world developing countries. In fact, open source advocates have already become a somewhat serious player in the fight for the ownership of the knowledge economy.⁴²⁷

6.3 Conclusion: IPR Laws Can Be Tuned

What could be done better to minimize the social costs of IPR infringement risk for open source in the future? From a company perspective, the insurance option is worth considering. The annual price of the insurance reflects the actual price of software patents. Also other IPR risk management techniques can be developed further from the level of source code integrity up until political activity.

From society's perspective, the picture is more blurred. Regarding patents, optimists such as Jaffe and Lerner argue that the obvious problems can be solved by improving patent quality and decreasing incentives for harmful patent suits. One can also take a more skeptical position if one does not buy the argument that the patenting system works for software innovations in the first place. A more radical solution would be to alter the property rules under patent law. First step would be to include exceptions in the coverage of the patent law.

In the long term, one can further argue, the liability rules in intellectual property rights legislation could be revised. The mere existence of socially undesirable accidents, as we have called them, is based on the *strict liability rule* for infringements defined in intellectual property laws. One could

⁴²⁷ For a recent overview of the issues, see Drahos and Braithwaite (2002)

think of the introduction of negligence standard for situations where the developers can't realistically know whether their software violates third party intellectual property rights. In fact, such a reform wouldn't be a radical departure from the current law. For example, during the national implementation of European Union Copyright Directive in Finland it was stated that anyone who downloads copyrighted works from the Internet is not liable if he didn't know – based on negligence standard – the material is infringing.⁴²⁸

⁴²⁸ See Memorandum of the Constitutional Law Committee of the Finnish Parliament (2005)

7 OFFENSE WITH OPEN SOURCE: CASE STUDIES ON LICENSING

This chapter studies how open source licenses have been used offensively as a part of market-changing business strategy. First case study is about operating system software markets and how open source alternatives have changed the existing market structure during the recent years. The second one describes a specific open source licensing model called dual licensing and discusses how several start-up companies have benefited from using it.

7.1 Licensing Open Source for Profit

7.1.1 Product Pricing Possibilities

Open source limits the possibilities of software pricing but does not make it completely impossible. Of course, already discussed indirect means to price copyrighted works are possible.⁴²⁹ Indirect pricing is especially suitable for software service businesses. Direct licensing to many users with open source is usually not feasible taking into account the “no royalty” requirement of Open Source Definition. However, with some imagination, at least the following direct pricing options are possible:

1. *First sale.* A license becomes applicable only after a first copy of the program is distributed. Thus, it is possible to charge price for the first distribution. This pricing model is applied for example in customized software projects.
2. *Renting over a network.* Since open source licenses apply only to software distributions and not to software use per se, any modified open source software that can be used over a network can be kept secret and in-house. This pricing model could be applied to software that

⁴²⁹ See section 3.2.5

can be used over the Internet, such as search engines and online marketplaces.⁴³⁰

3. *Dual licensing.* For the copyright holder, it is possible to release the same software package with different license terms. Those users who may not want to be bind by an open source license, can then purchase another (proprietary) license. This option is attractive to especially embedded software products with commercial applications. Adding new proprietary features to an open source base product is also close to this pricing model.

In addition to copyright, direct pricing based on other intellectual property rights may be possible. Both trademark and patent licensing may be available although it must be kept in mind that at the same time fulfilling the requirements of free copying and distribution substantially limits the possibility to effectively use them. In practice, for example trademark may be more valuable as supporting indirect licensing models and preventing forking to some extent.⁴³¹

7.1.2 Problem of Development Control

Open source poses also challenges to the coordination of software development, which is typically a necessary prerequisite for direct pricing. Specifically, development can't be coordinated with the means of copyright or secret source code. Instead, alternative means to appropriate software innovation come into play. Development control requires strong technical project lead with open opportunity for skilled programmers to participate. It is always possible for a project to separate into different paths if some developers or users are unhappy with e.g. the technical direction, project management or even licensing issues of a particular project.⁴³²

⁴³⁰ Although some recent open source licenses explicitly try to disallow the use of this model.

⁴³¹ Rosen (2001).

⁴³² One recent example was when Xfree86 project on a free software implementation of the X windowing system – key component in Unix based operating systems including Linux – changed its licensing policy towards more restricted in early 2004. It didn't take long for a new fork X.org to take the development lead. See Wheeler (2004).

In practice, many developers consider it central to the idea of open source that licenses allow the development of competing products (forks) although it is considered an ultimate response to a development lock-up.⁴³³ Also for companies operating in typical open source environments such as the Internet infrastructure, the avoidance of forking may be crucial for sustainable business.

An example is SSH Communications Security Corp, a Finnish start-up, which was listed in 2000 at the end of the dot-com boom at the Helsinki Stock Exchange with quick growth history. The history of SSH goes back to the early 1990s when Tatu Ylönen developed a secure shell protocol, named SSH, which eventually became a *de facto* Internet standard.⁴³⁴ The license terms of the first SSH were essentially open source stating: "As far as I am concerned, the code I have written for this software can be used freely for any purpose." In 1995, Ylönen founded a company for the commercialization of SSH and changed its license terms to proprietary. He never again released a new SSH version under an open source license.

Later, as the demand for SSH kept on growing, an open source fork emerged. While SSH Communication Security Corp did have more liberal license terms for non-commercial users (such as universities) it wasn't enough for those free software enthusiasts who wanted to embed SSH into their projects without any royalty or other intellectual property concerns. OpenBSD project located the early source codes with the liberal license terms and eventually released a fork called OpenSSH in late 1999.⁴³⁵

The project rewrote all parts of SSH, which had third party licensing issues or potentially violated software patents and cryptographic

⁴³³ For example Raymond (2001) explains that "Splits in major projects have been rare, and always accompanied by re-labeling and a large volume of public self-justification. It is clear that, in such cases ... the splitters felt they were going against a fairly powerful community norm."

⁴³⁴ SSH was also standardized through IETF.

⁴³⁵ The project website informs us that, "OpenSSH is a derivative of the original free ssh 1.2.12 release from Tatu Ylönen. This version was the last one which was free enough for reuse by our project."

export control laws.⁴³⁶ OpenSSH concludes the history page in a revealing statement: "SSH. Completely free at last." It took around a year for OpenSSH to get more popular. In the beginning of 2001, SSH Communication Security Corp threatened to initiate a trademark dispute against the project to change its name.⁴³⁷

Now there is both a commercial and open source version of SSH available. However, SSH Communication Security Corp does not control the development of the free version, which has by now surpassed the commercialized original in popularity. The commercial version has experienced a sharp loss in market share:

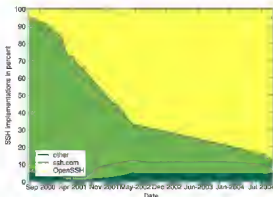


Figure 19 SSH usage on Internet servers.⁴³⁸

⁴³⁶ Original SSH license mentioned two software patents (called RSA and IDEA).

⁴³⁷ See OpenSSH (2001). Ylonen also made a somewhat misguided comment: "While free software is often good, it is usually not acceptable in the commercial world for critical systems (and I don't think that is going to change for many years)." The community response to Ylonen and his company was – unsurprisingly – negative.

⁴³⁸ See ScanSSH (2004) for continuously updated SSH usage statistics. It remains a speculation for the reader to consider what would have happened if the company had decided to continue to support the development of an open source version from the beginning and never let a fork to emerge.

7.2 Case Study 1: Free Licenses and Operating System Software⁴³⁹

The first case study describes the impact of different open source copyright licensing arrangements to the competition in microcomputer operating system markets. We compare the licensing policies of Microsoft Windows, Apple Os X and GNU/Linux operating systems. We argue that open source and free software have been the most important changing factors in the microcomputer operating system markets in the recent years: they has brought new entrants to relatively closed markets and changed the business model of incumbents. However, there has been no single open source strategy but merely all market players have adopted open source into their platform strategy in one form or other.

7.2.1 Introduction

Operating system software separated as a product from hardware in the 1970s. First popular interoperable operating systems were Unix systems started at AT&T in 1971. However, in the 1970s and 1980s Unix systems were not offered for microcomputers. First microcomputers in the late 1970s carried their own proprietary systems.⁴⁴⁰

In the early 1980s, there were basically two types on operating system markets on microcomputers: those controlled by hardware manufacturer such as Apple, Commodore and Atari, and new uncontrolled markets based on PC hardware standard. Open hardware on PC meant that there was in principle no single manufacturer control on operating system software. Soon, however, *de facto* operating system standard on PC hardware became Microsoft, first with Microsoft Disk Operating System (MS-DOS) and later with Microsoft Windows.

With computing evolving into new directions, Microsoft's operating system monopoly on PC hardware became challenged from time to time with the introduction of e.g. graphical user interfaces and networking features. But Microsoft resisted.

⁴³⁹ This case study is based on Välimäki and Oksanen (2005).

⁴⁴⁰ Campbell-Kelly (2003)

In the late 1990s, Unix compatible systems were finally offered for PCs. Linux and different Berkeley Software Distribution (BSD) variants combined with GNU system software started to gain popularity on cheap PC hardware and Internet servers. According to West and Dedrick, this new major trend towards open source was influenced by a need for cheap and free Unix implementations, the rise of different philosophy towards software ownership, and the rise of Internet as a new development and marketing platform.⁴⁴¹ Also proprietary mainframe and workstation Unixes were scaled down to run on cheaper hardware.

In the early 2000s, the operating system competition on the marketplace is again different. Microsoft Windows is still on the lead but competing especially on desktop markets with the new Apple OS X based on open source Unix compatible kernel. On server markets, new competition comes from a variation of highly developed free Unix implementations (GNU/Linux in the lead and different variants of BSD also available). Market leader Microsoft has been challenged on both fronts by either fully or partly open source software products.

This case study proceeds as follows. First we shortly overview the operating system markets on microcomputers as they stand today. Then, we discuss the historical development of different operating system licensing models and their implications to business model possibilities of Microsoft Windows, Apple OS X and GNU/Linux.⁴⁴² We show that there has been no single open source strategy but merely all market players have adopted open source into their operating system strategy in one form or other. Finally, we conclude the study arguing that free software and open source components have had a major impact on the microcomputer operating system markets in the recent years.

⁴⁴¹ West and Dedrick (2001)

⁴⁴² To simplify the analysis, proprietary Unixes such as Sun Solaris, HP/UX and IBM/AIX as well as fully open source BSD Unixes such as Free BSD were omitted. – Sun could have been worth a separate case study, however, as they announced in the midst of market pressures to release Solaris as open source in early 2005

7.2.2 Market Overview

The table below illustrates main operating system options available in the early 2000s.

	Windows	OS X	GNU/Linux
Provider	Microsoft	Apple	Many vendors
License(s)	Proprietary	Proprietary / Free	GPL
Market share	95-98 %	1-3 %	0.2 - 0.4 %
Main income	Licenses	Hardware, licenses	Support, services
Development	Inhouse	Inhouse / open source	Open source / community
Standards	Proprietary / open	Proprietary / open	Open
Processor	One option (x86)	One option (PowerPC)	Various
Environment	Desktop, server	Desktop, server	Server, desktop
Other hardware	Open	Proprietary	Open

Table 12 *Microcomputer operating system competition in the early 2000s*

A few notes of the table are in place. First, market share estimates need certain reservations. They are based on desktop use and indicate only operating system popularity on desktop computers. They are a poor indicator of the overall operating system market size and value consisting of both desktops and servers.⁴⁴³

Second, there is no single source of revenues. Microsoft can be perhaps best described as a traditional software company in the sense their main income comes from license sales. Apple, on their part, still builds on the hardware controlled operating system model: OS X runs only on Apple hardware and in order to run OS X, one needs to first buy an Apple computer. Finally, GNU/Linux systems are typically not sold separately at all but rather installed as part of larger computer investment. Revenues are generated indirectly from e.g. services and support.⁴⁴⁴

⁴⁴³ For the sources of these figures, see section 2.1.3.

⁴⁴⁴ Raymond (2001).

7.2.3 Study Framework

In order to understand more profoundly the impact of open source and free software licensing on operating systems competition, the licensing models of the three major operating systems were analyzed in more detail. The study was made by first collecting operating system market information from trade literature, vendor websites and expert interviews available on the Internet. Then, this information was analyzed through relevant economic theory. The presentation of each operating system is divided in three parts: historical background, licensing model, and impact on competition.

The first question is the licensing and implied ownership structure of the operating system software. Three main options used are proprietary (one owns), reciprocal (no one owns) and permissive licensing (everyone owns). GNU GPL is a typical reciprocal license while BSD and Apple Public Source License are more permissive. An overview of licensing options used at the different level of the system is illustrated in the table 13 below. These licensing options and their implications are explained later in detail with each operating system.

	Proprietary	Permissive	Reciprocal
Kernel	Microsoft	Apple	Linux
User interface	Microsoft, Apple	Linux	
Applications	All	All	All

Table 13. Main licensing options in different logical levels of the operating system.

Second, we discuss different business model alternatives. Possibilities for operating system vendors to benefit from components strategy include bundled proprietary software such as user interface, hardware and operating system integration. It is also possible to get involved in the complementary products and services markets including participation in open source communities. We assume business model possibilities depend on e.g. the licensing choice and whether the operating system is competing on desktop or server and enterprise markets.

Third, we discuss the competitive environment of each operating system. How system vendors have differentiated their product from others? What are their competitive advantages? We are especially interested how open source licensing has affected the competitive environment and has it been used as a competitive tool.

7.2.4 Microsoft Windows

Historical background. The first version of Microsoft Disk Operating System (MS-DOS) was released with IBM PC back in 1981. The first version of Microsoft Windows – at first basically a graphical user interface extension to MS-DOS – was announced in 1983 and published in 1985. From the beginning, Microsoft's operating system has been sold as licensed software. It has been priced as a consumer product being roughly 5-10% of the total price of a home PC system.⁴⁶

As graphical user interfaces became standardized Microsoft had a MS-DOS based edge on PC systems. Its many competitors including IBM's OS/2, Digital Research's Graphical Environment Manager (GEM), Berkeley Softworks' GeoWorks (GEOS) disappeared by the 1990s. While the competitors were perhaps technologically more advanced at the time compared to Windows they were more or less incompatible with MS-DOS programs. IBM's OS/2 offered the toughest competition. It was in the beginning supported by Microsoft but it never gained enough popularity to attract application developers, which were already developing for the growing Windows markets.

Licensing model and open source. Microsoft's licensing model has developed during time. From the beginning, Microsoft has been the driving force of mass-market software licensing. The company has lead the introduction of cheap and relatively simple end-user licenses to business software. Later on, the company has developed pricing strategies and price discrimination. Their licenses are tied to e.g. software itself (number of copies), user groups (students, home users) or hardware (OEM sales).

⁴⁶ See Computer Hope (2003) for historical Windows licensing prices.

With the increasing commoditization of software Microsoft has bundled other office, Internet and multimedia application to its operating system. From economic perspective, Microsoft has been a text-book example of how to use network effects to benefit from increasing returns and create a strong vendor lock-in.⁴⁴⁶ No wonder, antitrust officials have followed closely on Microsoft's licensing and bundling practices and the company has been the target of several unsuccessful suits.⁴⁴⁷

Recently, the company has changed its enterprise licensing model effectively from non-perpetual licenses to annual renewal fees. This "Microsoft Software Assurance" introduced three-year licensing cycles. Critics claimed that the company uses their lock-in power to force the users to upgrade their systems more often than really needed.⁴⁴⁸

In addition, Microsoft has to some degree opened Windows source code to its customers. Bigger customers and especially governments are hesitant to use products, whose source code they cannot inspect for security bugs and possible backdoors. Therefore, Microsoft introduced "Shared Source Initiative", which opened the Windows source code for inspections under strict non-disclosure agreements. Unlike in normal open source, the users were not allowed to make any changes to the source code or use it in their own products.⁴⁴⁹

Microsoft has subsequently gradually relaxed the licensing terms of at least part of its products. For example, the Shared Source License for Microsoft ASP.NET Starter Kit would almost pass the open source definition – the users are allowed to create derivate works and distribute them as long as they keep the original license agreement intact. A significant restriction, however, is that the users are not allowed the mix the source code with other code under a reciprocal open source license:⁴⁵⁰

⁴⁴⁶ Cf. Liebowitz and Margolis (2001) who criticize the blind application of network economics in the support of Microsoft antitrust claims

⁴⁴⁷ Quite descriptively, the US Department of Justice maintains a webpage titled "US v. Microsoft – the current case" [emphasis added]. See US vs. Microsoft (2003)

⁴⁴⁸ See Microsoft (2003c) for details. The original deadline for signing up for the program had to be extended because of the criticism. See ComputerWire (2002)

⁴⁴⁹ Microsoft (2003b).

⁴⁵⁰ *ibid*

"That you are not allowed to combine or distribute the Software with other software that is licensed under terms that seek to require that the Software (or any intellectual property in it) be provided in source code form, licensed to others to allow the creation or distribution of derivative works, or distributed without charge."

Overall, Microsoft's position on open source has changed considerably. Currently the company's position on open source is still somewhat critical and dismissive but not totally hostile.⁴⁵¹

"...The main benefit of the OSS model is that it allows any programmer to advance the ideas of the original developer, and global "communities" of programmers do emerge to contribute to major OSS projects. Another obvious benefit is that there is little or no cost in obtaining OSS software, although training, service, and support costs may be higher over the life of the software. The principal drawback of OSS is that no single entity can be held responsible for individual contributions of a far-flung army of unrelated programmers. There also is the possibility that one version of an OSS program will not work properly, or at all, with other versions. In addition, it is not clear that the OSS model can sustain software companies over the long term."

However, Microsoft still makes a strict difference between GNU GPL and other open source licenses. The company has used considerable resources to lobby against the use of GNU GPL in governments and publicly funded research arguing that GPL threatens the "software ecosystem".⁴⁵² As long as open source supports and complements Microsoft's product business and does not threaten the licensing business of its core intellectual property, this seems to be a rational policy in accordance with economic theory.

⁴⁵¹ *ibid.*

⁴⁵² See section 2.4.5 and Microsoft (2003a)

Impact on competition. Because of its current dominant market position (especially on desktop), it is reasonable to assume that Microsoft can only lose market share. Thus its competitive moves on operating system markets are more or less defensive. For example, the adoption of limited shared source programs is clearly aimed to undermine the threat presented by open source.

Microsoft has also used other strategies. The infamous "Halloween document" – a leaked Microsoft's internal memo on open source – draws an interesting (though painted) picture of their options.⁴⁵³ First, Microsoft can use its market power to divert standards to include proprietary components. This behavior is sometimes called as "extend and embrace". One well-documented example of this strategy is Kerberos-protocol, to which Microsoft made subtle changes to prevent Unix-servers to interoperate with Windows-clients.⁴⁵⁴ Of course, this strategy assumes that Microsoft currently controls some essential operating system component that can be changed without notifying competitors

Second possible strategy is aggressive use of intellectual property rights and especially software patents. However, it is worth noting that companies with large patent portfolios including IBM, which has the biggest patent portfolio of all, have now major business interest in GNU/Linux. They could and most likely would counter any move from Microsoft against GNU/Linux with similar or bigger charge of patent violations. And as noted in the previous chapter, some of the biggest companies – IBM including – have already pledged some of their patents to open source developers.

Microsoft's has also invested in trusted computing.⁴⁵⁵ Trusted computing protects files with hardware authentication thus making it in principle possible to control what the users install on their systems. As noted earlier

⁴⁵³ It must be noted that the Halloween document was circulated by open source enthusiasts with considerably critical views on Microsoft

⁴⁵⁴ Livingston (2000)

⁴⁵⁵ The company uses currently term Trustworthy Computing. At the other extreme of the naming game, Free Software Foundation prefers term Treacherous Computing

in this book, trusted computing is a potential threat to open source developers although so far none of the alleged risks have realized.⁴⁵⁶

In fact, even though Microsoft could theoretically act against open source in many ways, any such move could end up being counter-beneficial. The users do not want to lock themselves up to one vendor if there is an alternative. As open source product get more support from the industry, it is practically impossible for one company to try to block open source from the markets. Thus, Microsoft has been slowly turning from a critical bystander to cautious participant.⁴⁵⁷

7.2.5 Apple OS X

Historical background. Apple Macintosh had a strong position in microcomputer markets in the 1980s. However, in the 1990s company run into troubles and much was due to its ancient operating system technology. After reviewing the possibilities, Apple acquired NeXT, a company headed by Apple's co-founder Steve Jobs. A completely new operating system called Mac OS X was developed with Jobs' lead in the late 1990s.⁴⁵⁸

In short, Apple OS X can be described as a further developed version of NeXT's NeXTStep operating system with new graphical user interface and compatibility with old Mac software. OS X's kernel and other important parts – following NeXTStep's design – are largely based on BSD Unix and are open source. BSD itself has never been that popular operating system but its legacy lives on. In 1993, BSD was ported to cheap Intel hardware by William Jolitz and later three separate projects – FreeBSD, NetBSD and OpenBSD – were founded to continue BSD development. However, Apple's Mac OS X is by all accounts the first popular BSD implementation for microcomputers.⁴⁵⁹

Licensing model and open source. The user interface and many other system tools related to OS X technology are proprietary. Thus, the whole

⁴⁵⁶ See section 4.4.4 above.

⁴⁵⁷ Microsoft published its first two open source projects (not related to Shared Source) in 2004. See Jo Foley (2004).

⁴⁵⁸ West (2003).

⁴⁵⁹ See Howard (2001) and Hubbard (2003).

OS X operating system package is licensed much the same way as Microsoft Windows. There are no licensing requirements for applications running on Mac OS X. A traditional Mac software culture has perhaps favored proprietary licensing such as shareware for hobbyist programs. However, OS X has provided access to the system for many GNU/Linux open source programmers.

Because of the BSD license Apple has been able to also re-license everything in Mac OS X kernel with its own terms. The kernel, called Darwin, uses Apple Public Source License, which is accepted as free software license by the Free Software Foundation. Their approach has been to keep the source code of the kernel open and have good contacts with the open source community. For example, Apple has hired key developers including Jordan Hubbard, who is one FreeBSD founders and was a long time a core member of the development group. At the moment, Apple says it uses FreeBSD as a reference operating system. Apple also makes its modifications publicly available for the community even if there is no such license requirement. According to Darwin FAQ, the reason is:⁴⁶⁰

"Although the BSD licenses don't require companies to post their sources, divergent code bases are very hard to maintain. We believe that the open source model is the most effective form of development for certain types of software. By pooling our expertise with the open source development community, we expect to improve the quality, performance, and feature set of our software."

Impact on competition. Theoretically Apple's approach takes the best of all possible worlds. The open sourced Darwin and good community relations allowed Apple to leverage off the large community networks around different BSD variants. Good interoperability with Unix allowed also easy porting from rich collections of server applications. Open source code has helped hardware vendors to port their drivers to Apple. Also, Apple has worked in collaboration with Microsoft and most of the proprietary Microsoft-controlled standards are supported in OS X (file formats, Internet ex-

⁴⁶⁰ Apple (2003).

tensions). Finally, and perhaps also most importantly, Apple held control over the OS X user interface, which allowed it to differentiate the product from the rest of the Unix variants.⁴⁶¹

Unfortunately, the market realities have somewhat spoiled Apple's approach. Even if Darwin has been ported to Intel-based hardware, without user interface it is more or less useless. This has limited the developer base to the core Macintosh users, which had already the hardware from Apple. This most likely also explains why Darwin has not gained similar momentum as Linux did. It is also important to note that Apple cannot benefit directly from the Linux-development because it is not possible to attach GPL'ed software to Darwin.

7.2.6 GNU/Linux Distributions

Historical background. The history of both GNU and Linux are by now well known and easily available on the Internet. On the one side, we have Richard Stallman, Free Software Foundation and the GNU GPL, first implemented in 1989. On the other side we have Linus Torvalds and Linux.

Torvalds had started to develop his new operating system in 1991. Linux was the first Unix implementation targeted for microcomputers. In 1992, Torvalds decided to license Linux under GNU GPL and subsequently all source code written to the Linux kernel by numerous contributors was under that license. Most system tools of Linux were taken from GNU project and other sources including BSD. Linux itself became the kernel of the new operating system.

Later, Free Software Foundation and Stallman have required the full operating system based on main components from the GNU project to be called GNU/Linux. While perhaps technically applicable, trade press has so far neglected Stallman's desire. In this thesis, however, we follow Stallman's recommendation in order to emphasize difference components of an operating system (kernel, user interface, system tools etc.).

⁴⁶¹ West (2003)

Licensing model and open source. Most software in GNU/Linux distributions are under GNU GPL license. The license states that the software must be free to copy, distribute and modify. Also, the source code must be open and available practically free of charge. It is not allowed to use the GNU GPL copyrighted code in derivative works (as defined in law) under any other license but GNU GPL. The interpretation of the last mentioned term is unfortunately quite open.

GNU/Linux includes also essential system software licensed with other terms. Free Software Foundation has for example used other license terms for their programming language compilers. The idea is that any new program created in GNU/Linux – being perhaps technically speaking a derivative of the compiler – is not automatically under GNU GPL. The current interpretation of GNU GPL and derivative works means that proprietary applications and system tools can be developed, run and even sold with proprietary licenses for Linux. However, the rough rule is that it must be possible to run any such application separately (using only dynamic runtime-calls) from other program including the kernel.⁴⁶²

A major part of GNU/Linux is X11 windowing system based user interfaces. X11 is licensed with liberal BSD-like terms but the most popular graphical user interfaces including KDE and Gnome are under GNU GPL. Major problems in their development have been usability issues as well as the lack of open source GUI code. Developing a major free software operating system component from scratch has proven to be a time consuming project.

Impact on competition. In a way, GNU/Linux is not a ready-to-run product. It is a set of commodity tools and components that can be used as such and tailored for different uses. There are numerous companies distributing or otherwise using GNU/Linux under different business models such as Red Hat and Novell for desktops and servers, and MontaVista for embedded systems. Large software companies including IBM, Oracle and SUN support Linux development and use it to sell their proprietary enter-

⁴⁶² Free Software Foundation (2003)

prise application software running on Linux.⁴⁶³ For example, SUN states on their website:

"Seven Sun ONE products are available on Linux today - Application server, Directory server, Web server, Active Server Pages, Studio, Grid Engine and Message Queue - with plans to deliver more in the near future."

In a way, GNU/Linux works as an open network providing opportunities to companies that sell related and interoperable products or services to the network members. Therefore, it is perhaps more appropriate not to take the GNU/Linux as a direct competitor to Microsoft Windows or Apple Mac OS X. First, GNU/Linux distributions have a minimal market share on desktop users. In addition, Microsoft has been hesitant to support Linux and all kinds of interoperability problems with the dominant operating system have made consumer markets difficult. Second, on servers a large market share is captured by enterprise application providers, which use GNU/Linux as an independent and cheap platform for their applications. These applications can be also run on different operating system's including provider's own (e.g. IBM AIX and SUN Solaris).

7.2.7 Concluding Remarks

From historical perspective, it seems clear that open source licensing has indeed changed the operating system software markets in the recent years. The fact is that today the main operating system alternatives are either fully or partly open source Unices or Microsoft Windows. Ten years ago, Microsoft had gained practical monopoly on microcomputer operating systems with only Apple Macintosh barely hanging along. As of today, both Apple and the new entrant GNU/Linux hold stronger position largely thanks to open source development methods and the rapid growth of Internet and networked computing.

⁴⁶³ See e.g. Linux at IBM (<http://www.ibm.com/linux/>) and Linux from Sun (<http://www.sun.com/software/linux/>)

However, there seems to be no “one-size-fits-all” open source operating system strategy. The open community development model guaranteed fast growth for GNU/Linux. Anyone can download, modify, and distribute the source code of the whole operating system free of charge. To compare, Apple’s Mac OS X has only an open source kernel with an in-house development model controlled by Apple. Other parts of the operating system including the user interface are proprietary and the whole system is licensed for fee. Finally, Microsoft’s reply to open source has been their much debated shared source initiative. Significant institutional users have been granted under specific shared source agreements a limited access to view parts of Windows source code. Microsoft basically still believes in the fully proprietary development and licensing model.

Therefore, it may be difficult to understand how open source actually works as a competitive tool. Much remains for further study. The little data we have presented in this section can be perhaps summarized as follows:

- Open source code and free software have proved to be powerful ways to standardize and stabilize new operating system technology and compete against established market powers
- On desktop markets, their impact has been limited mainly because of compatibility and usability issues (strong lock-in to incumbent operating system vendors)
- On server and enterprise application markets, they have had more changing impact based on the benefits of standardized independent technology and other technical features (weaker lock-in to incumbents)
- Proprietary components are still a major competitive factor and there remains large areas in both desktops (e.g. user interface) and servers (e.g. enterprise applications) without major open source impact from where proprietary vendors can generate revenue

7.3 Case Study 2: Dual Licensing and Embedded Software⁴⁶⁴

We noted in previous chapters that developers sometimes dual license their projects because of license incompatibility issues. In this section, we analyze how dual licensing can be used as a business model. The evolution and functionality of the dual licensing model is explained through three case studies: Sleepycat Software Inc, TrollTech AS, and MySQL AB. Each of these companies is an open source start-up that has been able to build a profitable business based on the licensing model. At the end of the section, legal requirements, economic implications and practical limitations of dual licensing are discussed.

7.3.1 How Dual Licensing Works?

Dual licensing mixes open source and proprietary business models. Duality means that both the free software distribution mechanism and traditional software product business are combined. There is technically only one core product but two licenses: one for free distribution and free use and another for other (proprietary) uses.

Dual licensing model differs from pure free software model in several ways. First, the development community does not have development power to start competing products (forks). Copyright and control of the core product development is held in one hand, the original developer. The ability to license the product with other terms than open source requires full ownership of all rights to the product.

Second, the users of the free license have an option to obtain a proprietary license. If a software product with reciprocity obligation – as for example term 2b) in the GNU GPL – is embedded to become a part of another product then the combined product should be distributed for free. A proprietary license may free the user from this restriction. In this way,

⁴⁶⁴ An earlier version of this case study was published in Valimäki (2003b). The author wishes to thank Michael Olsen of Sleepycat Software Inc, Torge Sund of TrollTech AS and Mårten Mickos of MySQL AB for kindly providing information on their companies and the participants of Free Software Business mailing list fsb@crynwi.com for fruitful email exchange discussing an earlier version of this case study.

third party product businesses become also possible. From the user's perspective, dual licensing can be described as indiscriminating.

Figure 20 describes the dual licensing model in more detail.

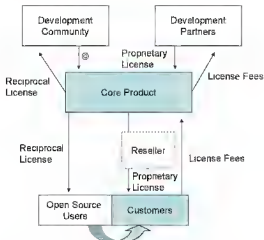


Figure 20. License streams of a core product in a simplified dual licensing model.

Let's look at the figure 1 from the bottom up. In the bottom are software users divided into two segments. The dual licensed software (core product) is both licensed with reciprocity obligation to first user segment titled "open source users" and with a commercial license to another user segment titled "customers". The arrow from open source users to customers indicates that when the open source users extend the usage of the software they tend to reach the limits of free use. For example reciprocity concerns, commercial support, warranty requirement or similar reasons may attract them to buy a commercial (proprietary) license.

Above the core product there are two developer segments. On the left is the open source developer community, which may give bug fixes and code contributions with copyright back to the core product developers. On the

right are commercial development partners which develop essential components of the core product; they may either transfer or license (perhaps more common) the copyright of the component to core developers.

7.3.2 Study Framework

Three open source companies using dual licensing model were selected for detailed analysis: Sleepycat Software Inc, TrollTech AS, and MySQL AB. These companies were selected mainly because they are the first successful, well-known and long-term users of the dual licensing model.⁴⁵⁵ The study was made by collecting information from company websites, referring to company executive interviews available on the Internet and by asking complementary questions directly from company executives. The presentation of each company is divided in three parts: historical background, licensing model and model effectiveness.

The first question is how the companies ended up in a dual licensing model. It turns out that none of the companies have started from a dual licensing model. The concept has evolved as time has passed and both the Internet and open source markets (especially based on Linux) have matured. The second question is how the dual licensing model works in each case. What licenses do the companies use and for what reasons? Here we learn that while the companies may use different free software licenses they all contain a strong reciprocity obligation. Finally, we consider the effectiveness of the dual licensing model in each case. How do the companies benefit from the free use of their software compared to the traditional software publishing model? Have the companies detected a piracy problem? How do they manage the legal rights in the products they own?

⁴⁵⁵ There are many more recently started companies, which aim to benefit from the model. Also some big company ventures experiment with the model: e.g. Sun Microsystems uses dual licensing when they sell Star Office – a proprietary version of Open Office.

	Sleepycat Software Inc	MySQL AB	TrollTech AS
Product	Embedded database	SQL database	GUI tools
Free license(s)	Sleepycat License	GPL	GPL, QPL
Users	Millions	Approx. 4 million users	Hundreds of thousands
Customers	Thousands	Around 0.1 % of users	Thousands
Main income	Licenses, services, support	Licenses, brand, services	Licenses
Development	Inhouse	Inhouse	Inhouse
Marketing	Direct and indirect	Direct	Direct and indirect
Technology	Standardized (database)	Standardized (SQL)	Non-standard (GUI)

Table 14 Some attributes of the studied open source products.

7.3.3 Sleepycat Software Inc.

Background. Sleepycat Software Inc. develops and markets BerkeleyDB (BDB). The product is an embedded database system that runs on multiple platforms. The first version of BDB was written by Keith Bostic and Margo Seltzer in 1991. It was released under BSD license as part of BSD Unix distribution from University of California at Berkeley. BDB was distributed freely on the Internet and eventually many open source as well as proprietary projects started using the product. BSD license terms allowed a wide adoption of the product even in commercial projects with no license fees to copyright owners.

As the product gained more commercial interest, the programmers decided to found the company Sleepycat Software Inc. as the owner of the copyright and develop the product further. The next version added technical functionality and was therefore commercially even more valuable. It was released under Sleepycat License in 1997. From then on, BDB has been licensed under a dual licensing model.⁴⁶⁶

Licensing model. Sleepycat's website states on their licensing policy:⁴⁶⁷

"The Sleepycat open source license permits you to use Berkeley DB ... at no charge under the condition that if you use the software

⁴⁶⁶ Zimran (2001).

⁴⁶⁷ Sleepycat (2004).

in an application you redistribute, the complete source code for your application must be available and freely redistributable under reasonable conditions. If you do not want to release the source code for your application, you may purchase a license from Sleepycat Software.”

Sleepycat’s CEO Michael Olsen has described the usage of the proprietary license in an interview:⁴⁶⁸

“If a company wants to redistribute Berkeley DB as a part of a proprietary product, they can come to Sleepycat and pay us a fee to purchase different license terms from us. In that case, we sign a pretty conventional license agreement permitting use and redistribution in binary form, without forcing them to ship source.”

Model effectiveness. The development of BDB is directed within the company. All outside contributions are implemented by company developers into the core product. Obviously, the development of a complex database engine requires understanding of the functionality of the whole. Development of add-on features is difficult. Therefore, user feedback benefits mainly in identifying bugs and proposing new features. It would be possible to change Sleepycat License into GNU GPL but at the moment there seems to be no immediate reason for this as the Sleepycat license has been widely accepted in the free software community.

Most of the income of Sleepycat (around 75%) comes from license sales and the rest from services. Sleepycat does not promote other than license sales on their website, which is their main direct marketing channel. The usage under free license is not monitored. If license breaches are found, which is not very common, the users either buy a proprietary license or stop using the product.⁴⁶⁹

⁴⁶⁸ As quoted in Zinman (2001).

⁴⁶⁹ Zinman (2001).

7.3.4 MySQL AB

Background. The product of MySQL AB is a relational database management system. It was first targeted at web server use but is now offered also as a general database management system and specifically to users of embedded databases. The development started in 1995 by Michael Widenius and David Axmark and the first major release on the Internet followed in 1996.

From the start, MySQL shipped with its own license terms. (MySQL 1995) That license allowed limited free distribution and usage of the product with a strong reciprocity obligation on Unix-based systems (including Linux). On Windows, the license model was shareware restricting the free use and distribution of the product. Their business model was essentially dual licensing on Unix-based systems and proprietary on Windows.

As the Linux-based version became very popular on the Internet, the free license was changed in 2000 into GNU GPL on all platforms. After that, their licensing model has been solely dual licensing. The license change limited the scope of proprietary licensing for different uses but at the same time it attracted even more users for the product. As late as in 2001, the company MySQL AB was founded to own the copyright to the database software with its partners.⁴⁷⁹

Licensing model. The product's copyright is licensed either by GNU GPL or a proprietary license. Products that include the GPLed version must be licensed under GPL. MySQL's website states their licensing policy in the following:⁴⁷¹

"If your software is licensed under either the GPL-compatible Free Software License as defined by the Free Software Foundation or approved by OSI, then use our GPL licensed version. If you distribute a proprietary application in any way, and you are not licensing and distributing your source code under GPL, you need to purchase a

⁴⁷⁹ Grant (2002)

⁴⁷¹ MySQL (2004a)

commercial license of MySQL. If you are unsure, we recommend that you buy our cost effective commercial licenses.”

Model effectiveness. Development is directed inside the company. As with Sleepycat, the product is very complex and can hardly be developed by third parties. In 2001, another company tried a fork but failed without being able to control the software development.⁴⁷³ All contributions are checked and rewritten by company developers thus not diluting the copy-right ownership of the product. MySQL currently includes one major component developed and licensed by a third party.⁴⁷³ The company estimates that they have fewer problems with free riders than proprietary software companies; the only case that has ended up in a court was with the fork.

As of today, MySQL AB receives more income from proprietary license sales than from their other income sources, branding and services. Their main income seems to come from embedded commercial users.⁴⁷⁴ To contrast, use on web sites – the product's initial market – seems to work after the license change to GPL as a marketing tool for commercially licensed use on embedded products.

7.3.5 TrollTech AS

Background. TrollTech AS's main product is called Qt, which consists essentially of graphical user interface programming libraries. Qt can be used to develop multi-platform graphical applications. As a result, the developed products embedded functionality from Qt libraries.

Development of Qt started in 1992 and the company was founded in 1994 by Haavard Nord and Eirik Eng. In 1996 Qt was released under its own quite restricted open source license, which did not allow free distribution of modifications and hence retained full development control with TrollTech. However, due to available source code Qt was selected to be

⁴⁷³ Progress Software Corp. v. MySQL AB (2002). See MySQL (2001) for more information on the dispute from MySQL AB's perspective.

⁴⁷⁴ See InnoDB (2004) for more details.

⁴⁷⁵ Codewalkers (2002).

used in KDE, which quickly became a very popular free desktop environment for Unix and Linux systems.

As its popularity and importance grew with KDE, pressure from the free software community to allow redistributable modification increased. In 1998 the license was changed to QPL, TrollTech's own reciprocal license. QPL allows distribution of modifications as separate patches. In 2000 Qt was finally released also under GNU GPL allowing modifications of the entire software to be distributed for free. The GPL release was also delayed by the company founder's initial skepticism towards open source.⁴⁷⁵

Licensing model. The licensing model of Qt is essentially the same as with the two products described above. Qt is licensed under GPL, QPL and a proprietary license. Products made with the GPLed (or QPLed) version must use the same free license. TrollTech's website states:⁴⁷⁶

"Based on the "Quid Pro Quo" principle, if you wish to derive a commercial advantage by not releasing your application under an open source license, you must purchase an appropriate number of commercial licenses from Trolltech. By purchasing commercial licenses, you are no longer obliged to publish your source code."

Model effectiveness. Development is coordinated within the company. Before the introduction of QPL, TrollTech's license terms granted that the company had full control of the development. However, now Qt also contains some code that is not owned by TrollTech AS but is rather licensed under a very permissive license from third parties.⁴⁷⁷ From licensing perspective, the introduction of GPL includes the possibility of forks but in practice it seemed to result instead in an excellent marketing move.⁴⁷⁸

TrollTech sales are based on proprietary licenses. Qt is marketed through a combination of direct sales, resellers, and strategic partners. The

⁴⁷⁵ Fremy (2001)

⁴⁷⁶ TrollTech (2004)

⁴⁷⁷ TrollTech (2003).

⁴⁷⁸ Fremy (2001)

role of the free version is mainly to grow the user base and market the product on the KDE environment.⁴⁷⁹

7.3.6 When Does Dual Licensing Make Sense?

Licensing and intellectual property issues. In every dual licensing case example, the open source license included a *strong reciprocity obligation*. As noted, strong reciprocity (also called copyleft) means that even adaptations and derivative works must keep the license terms intact. In other words, if the source code is initially distributed free of charge then no one can charge for the source code later in any adaptation. GNU GPL and Sleepy-cat Licenses are both accepted as copyleft by the Free Software Foundation.⁴⁸⁰ While QPL does not fill the strict community definition of copyleft, it also essentially functions in the same way.

Lerner and Tirole have argued that open source products targeted at developers tend to have permissive licenses.⁴⁸¹ The model presented in this case study contradicts their finding: in a dual licensing model, the software company uses a highly restrictive reciprocal license in a product specifically geared towards developers for embedded use. A possible explanation for this contradiction is that the data set studied by Lerner and Tirole consisted mainly of non-commercial projects at Sourceforge.

Another fundamental legal requirement for dual licensing is that the software company has *undisputed rights* to the software product it wishes to dual license. Ownership of rights is central because it allows company to price its software, change its licensing policy and distribute software with different licenses. A major legal risk in using open source licenses is that the license may dilute the ownership and even eliminate the possibility to dual license. Therefore, rights ownership must be managed carefully. The one who has written new or rewritten old software is granted exclusively copyright to the work. However, with multiple authors the copyright ownership may also become distributed. Under a strong reciprocity obligation, a fully open and distributed development process without suf-

⁴⁷⁹ Ibid.

⁴⁸⁰ Free Software Foundation (2004b)

⁴⁸¹ Lerner and Tirole (2005)

efficient rights clearing is not suitable for any company that wishes to make any direct license sales with dual licensing. No hidden liabilities in code contributions from unknown third parties should remain.

Economic issues. Dual licensing also depends on several distinctive economic implications that must be at hand. First, there must be sufficiently *large user base* for the product. Here, the reciprocal license enables strong network effects typical to information products: the value of the product to single user depends on the number of user it has. With the free availability and efficient distribution of the product through the Internet there are not many limitations for exponential user base growth. Especially software that depends on separate distributed components interoperating directly has strong network effect. Shy and Thisse have demonstrated that when network effects are strong, unrestricted copying and distribution of the software product results in an equilibrium in a simplified setting within non-cooperative competitors.⁴⁶²

Second, the effectiveness of dual licensing depends on *price discrimination*. A software company that manages all rights to the product may license it according to market demand.⁴⁶³ For example, our case study have shown that if there is demand for both stand-alone and embedded products, then dual licensing may be an economically viable model. Also worth noting is that in dual licensing the licensing policy – not the product features – are tailored. Only those users that have direct benefit from the use of the software are required to pay a license fee and for other users payment is more or less optional.

Third, there seems to be *no major requirements for the enforcement of copyright*. The little data we have indicates that high-end corporate users required to purchase a proprietary license also do so. Also empirical evidence shows that illegal copying of software in general has decreased during 1995-2000 relative to the software market.⁴⁶⁴ Open source license option may strengthen this trend. While traditional copyright licensing model is still plagued by a vast number of unauthorized users, the free li-

⁴⁶² Shy and Thisse (1999)

⁴⁶³ Shapiro and Varian (1999).

⁴⁶⁴ Osano (2002)

cense conversely supports free usage by the majority of users who would not pay the license fee anyway. Worth to note is also that for someone who embeds a free product into a commercial one, the license purchase does not lead to the ethical and philosophical issues one may have with traditional copyright enforcement and zero tolerance. Instead, dual licensing is one answer to the economic question of how the copyright owner should protect his work.

Finally, the case studies have shown that the company as an organization *needs to believe* in the reciprocal licensing model. At first, it may sound counterintuitive, but if the product has for example both stand-alone and embedded users then strong reciprocity may be workable. Moreover, GNU GPL license seems to be a viable marketing tool in dual licensing. TrollTech was not convinced that GNU GPL license would allow them to continue the sales of proprietary licenses until they tried. In side, they received increasing media attention and political acceptance. MySQL had almost similar experience.

7.4 Concluding Remarks

This chapter has used the developed economic and legal frameworks to analyze different actual licensing strategies through case studies. In the first part we discussed licensing strategies in operating system markets. In the second part we discussed dual licensing as a new business strategy.

The impact of open source licensing has been particularly clear and thorough in the markets for operating systems. Open source has forced incumbents to change their licensing strategies as the example of Linux has forced them to battle for developers and answer to increasing customer concerns and needs. Today, all major operating system software developers use open source licensing models at least in some part of their products.

Dual licensing – licensing technically identical product with both proprietary and open source licenses – seems to be a viable model for specific types of new software companies. However, dual licensing is no silver bullet. We have identified several organizational, legal and economic limita-

tions and requirements, which may in practice limit the usability of the model.

The general limitations of the case studies deserve also attention. Our case examples were few in number and discussed qualitatively mainly successful companies. For example, during the company selection process no particularly successful stand-alone end-user applications were encountered. Open source licensing strategies seem to work best for technical products that are aimed rather at developers than end-users. Operating systems and databases are exactly such products.

8 CONCLUSIONS

8.1 The Rise of Open Source

Open source has been part of the software industry from the start. The idea of open source code was however hiding during the 1980s and early 1990s when the development and market conditions favored centralized and closed development and proprietary licensing. In the 1990s, the environment then eventually changed.

We identified a number of explanations to open source's rise into the mainstream of the software industry. From *technical* perspective, the rapid growth of the Internet, the trend towards cheap PC computers and a need for more flexible development methods have supported open source. From *business* perspective, open source has offered possibilities for new companies to change the existing market structure and rules of the game. Open source has also altered the market strategies of incumbents. From *social policy* perspective, open source has been claimed to be a tool for more democracy, access to information and social equality as the role of software in the society continues to increase.

By any means, open source remains a source of controversies. There is no singular open source community but rather multiple individual voices stemming from the hacker culture, whose leftist and liberal ideals originate from the 1970s. Among software developers, Free Software Foundation and its supporters want software to be free from corporate control, while Open Source Initiative and other pragmatics only feel open source is the most effective way to innovate.

Also the software industry has debated about the benefits and drawbacks of open source for years now. Some companies, notably Microsoft, have openly criticized the principles of open source as anti intellectual property. Others openly support open source development – where it only benefits their strategy. After the founding of Open Source Initiative in 1998, open source software, development and licensing have arguably become more *corporate-friendly*. Ideological opinion leaders such as Richard

Stallman have somewhat marginalized and moved their attention from software development to more general social policy issues.

8.2 Impact on Licensing Practices

Journalist Robin “Roblumo” Miller describes the state of licensing discussion in early 2005 as follows:⁴⁸⁵

“A NewsForge article about software licensing or software patents two years ago would draw 20,000 - 40,000 readers and might lead to a Slashdot discussion with 500 comments, of which 150 were substantive. Now, although [open source software] is more widely used, the readership of licensing-related articles is typically less than 5000 (with rare exceptions), and Slashdot discussions on the topic are both smaller and less substantive.

I can't give you figures for other tech news sites, but I can tell you that, from what I read on IT journalism email lists, reportorial interest in software licensing issues is dropping steadily.”

Decreasing news interest in open source – and software licensing issues in general – suggests that the software industry has already learned what open source licensing is about. After all, the increasing use of open source hasn't changed licensing practices in the entire industry overnight. When open source licensing has indeed changed licensing practices, it has happened over a long period of time and in specific markets such as Internet infrastructure and operating systems.

From *legal perspective*, the use of standardized and relatively simple license templates could – in principle – mean less legal fuss. For instance, in joint software development projects the use of “neutral” open source licenses may provide a fair balance to the interests of all participants and save both time and money.

⁴⁸⁵ Robert “Roblumo” Miller's message to license-discuss@opensource.org March 3, 2005

However, the devil is in the details. At a closer look, open source licenses are not free from legal risks, ambiguous language and uncertain economic implications. The central question of reciprocity obligation – copyleft – is not settled, and the question even seems to be license-specific. The licenses have different approaches to software patents – for example, many licenses don't accept any patent royalties. Moreover, many open source licenses don't even accept the co-existence of other open source licenses being incompatible with each other.

Though the number of different open source licenses is already over fifty – and most of them are just confusing complex copies of others – there are only few truly popular licenses. Thus, despite all the fuss and uncertainties, the original ideas of simple, understandable and fair licensing practices still live. To compare, every proprietary software product typically carries its own, unique license with unique problems, and unique risks. Arguably the licensing issues of 1000 open source products are thus cheaper and more straightforward to manage than the licensing issues of 1000 proprietary products.

From *economic perspective*, the licenses seem to carry indeed fundamental implications. If direct license royalties – either based on copyright or patents – are not possible, how can a software company generate increasing returns? Do open source licenses “cannibalize” markets for proprietary licenses?

Perhaps they do in principle, but in practice market forces are already adapting. IPR restrictions are only one of the possible means to price software. As noted already in the first chapter, software is increasingly sold as services under long-term subscription agreements. Granted, also subscriptions may be based on copyright and patent laws. However, if they also bundle maintenance, support and other “services”, intellectual property royalties and software licenses are no longer that crucial. Moreover, we have witnessed open source start-ups that essentially use open source licenses as marketing tools for their proprietary offerings. To conclude, open source licensing should not be seen as an “either-or” dilemma but as an adaptive complement to the existing proprietary licensing practices in the software industry.

8.3 Impact on Intellectual Property Management

When intellectual property protection was initially granted for software products in the early 1980s the aim was to prevent free riding and foster the new software mass markets. The software copyright system soon balanced itself. Extreme proposals for copy protection systems didn't get through market forces and the copyrightability of interfaces was turned down by the courts and lawmakers. At the same time, however, the intellectual property system continued its expansion elsewhere: software patenting grew rapidly during the 1990s.

Legal scholars have continued to debate whether and to what extent copyright, patents or some other legal protection regime should be applied to computer programs. One of the main theoretical arguments for the protection of non-literal aspects of software is that software copyright is incomplete and thus insufficient. The argument goes that copyright does not cover what is really valuable in software. For example Samuelson et al explained in 1994 after interface trials that software copyright has met its limits and software needs *sui generis* protection – something like WIPO proposed already in the late 1970s adopting more elements from the patent system to copyright.⁴⁸⁶ Essentially similar arguments have been repeated recently by e.g. Lessig.⁴⁸⁷

The main weakness common to all of these arguments is their abstractness. It is difficult to give practical evidence that software markets based on copyright do not function or that the markets would function essentially better with additional intellectual property protection. Open source licensing is based essentially on copyright and supporting contract law provisions. Any additional intellectual property or technical protection would probably only endanger the functionality of the model.

Thus, it can be argued that open source highlights the *negative effects* from the continuous expansion of both intellectual property right legislation and the actual use of those rights. Especially increasing patenting has increased infringements risks in software development. Thus, intellectual

⁴⁸⁶ Samuelson et al (1994)

⁴⁸⁷ Lessig (2001)

property infringement risks must be taken more seriously at both company and community level. The fact is that no open source business model has so far benefited anything from patents and the expansion of intellectual property.

Licenses themselves are merely legal technique. However, they have changed thinking on *how to use copyright* and other intellectual property rights in practice. They have brought "Internet-businesses" into mainstream software markets. They have shown that it makes surprisingly often sense to give most exclusive rights to the general public for free. Obvious challenges remain. While the value of intellectual property probably increases from free sharing, it also becomes more complex to appropriate. The challenging business questions for any software company therefore is, how to navigate through open source licensing without losing business proposition.

8.4 Impact on Commercial Regulation and Legal Study

This book has showed how the software industry has started to adopt open source licensing models and changed their business propositions accordingly. New start-ups have challenged incumbents in such areas of software development where it was most unanticipated a few years back. At the moment, the principles of open source licensing are generalized to numerous other copyrighted works and patented inventions. There is vivid discussion and initiatives ranging from *open content* for music and movies to *open access* for science and educational material. "Open whatever" seems to be a kind of counterforce in the evolution towards *extensive commercial regulation of the society* at large.⁴⁸⁸

Thus, openness *balances* commercial regulation. When there was no extensive regulation of private entitlements to every imaginable object of the society, there simply was no need to balance the public interest by explicit anti-licensing. But today practically every little piece of software, musical compositions, moving pictures, scientific findings and educational litera-

⁴⁸⁸ Weber (2004), p. 267-268, warns not to overgeneralize open source, since many recent initiatives do not share "the property regime that makes open source distinctive"

ture is under more or less restrictive exclusive rights. For various economic, moral and social policy reasons explained in this study, there seems to be need for explicit licensing systems that realize the original balance of interests behind the exclusive rights of copyrights and patents, may it happen in the shadow of the law.⁴⁸⁹ In short, open licensing systems have proved how *overregulation can be fixed without state intervention*.

In this way, open source emphasizes a more *material* study of intellectual property rights. For sure, legal policy debate – with substantial input from economists, philosophers etc. – on the merits and drawbacks of new intellectual property treaties and legislation is already immense. Usually, the question is whether a proposed new extension to a given intellectual property right should be accepted. The study of open source and other licensing issues takes the focus from these borderline cases right back to the centre. When a substantial number of right holders in a given industry decide not to enforce their core intellectual property rights – relying on economic-rational arguments – the premises of the policy discussion can be seen in a new light. Why and how do companies do that? What does it mean to the intellectual property system as a whole? Whether one should hold to the government granted intellectual property rights to the fullest is again a relevant question for any software developer and lawmaker alike.

Can open source licensing research have any *impact on future (de)regulation*? This book has for example proposed reforms to intellectual property liability rules based on the needs and central role of open source developers in the society at large. Here, we must note that the political open source and intellectual property discussion is inherently global as the intellectual property policy is one of main issues at world trade negotiations. In the end these policies are human made. A promising note is that the number of discussing parties involved has increased in the relevant forums. For example different non-governmental interest groups have an increasing influence at the negotiations of WTO and WIPO.⁴⁹⁰ And especially

⁴⁸⁹ Ellickson (1991) noted that many social norms work “beyond the law” without any reference to actual legal rules. Meigs (2004) continues the language-game and points out that open source in fact works “despite the law” indirectly criticizing the potential threats from the expansion of intellectual property rights.

⁴⁹⁰ On the policy tendency in general see Castells (1996), pp. 80–90, and on intellectual property in particular Matthews (2000), pp. 128–135.

non-governmental organizations representing the interests of open source software developers have proposed that academic researcher should be taken more seriously into account when new policies are set and old ones revised.

FIGURES AND TABLES

FIGURE 1. TOTAL REVENUES FROM THE TOP 500 US-BASED SOFTWARE COMPANIES	15
FIGURE 2. MARKET SHARE OF WEB SERVERS.....	17
FIGURE 3. MAIN WEB SERVER SOFTWARE COMPONENTS IN THE EARLY 2000s	18
FIGURE 4. ILLUSTRATION OF LICENSING MODELS FROM REVENUE GENERATION PERSPECTIVE OVER TIME.	26
FIGURE 5. THREE MAIN WAYS TO DISTRIBUTE SOFTWARE PRODUCTS FROM SOURCE CODE PERSPECTIVE.....	29
FIGURE 6. BENEFITS OF OPEN SOURCE ACCORDING TO IT MANAGER INTERVIEWS IN 2004 . . .	38
FIGURE 7. CHALLENGES OF OPEN SOURCE ACCORDING TO IT MANAGER INTERVIEWS IN 2004.	39
FIGURE 8. DIFFERENT COST FUNCTIONS IN THE PRODUCTION OF INFORMATION GOODS	51
FIGURE 9. COMPONENTS APPROACH TO SOFTWARE PRODUCTS.	55
FIGURE 10. AN OPEN INNOVATION MODEL.	76
FIGURE 11. EVOLUTION OF SOFTWARE COPYRIGHT AND PATENTS IN THE INDUSTRY HISTORY.	108
FIGURE 12. FUNCTIONAL DIFFERENCES REGARDING COMBINATION AND MODIFICATION BETWEEN OPEN SOURCE LICENSES	119
FIGURE 13. ABSTRACTION-FILTRATION-COMPARISON METHOD	126
FIGURE 14. A SIMPLIFIED COMPONENT-BASED VIEW OF A COMPUTER PROGRAM.	127
FIGURE 15. DERIVATIVE WORKS AND LOADABLE MODULES	129
FIGURE 16. POSSIBLE TERMS WITH RELATED LOGOS IN CC-LICENSES.	156
FIGURE 17. DEVELOPER-CHAIN AND IPR INFRINGEMENT.....	166
FIGURE 18. US PATENTS GRANTED BETWEEN 1984-2004. SOURCE: USPTO	175
FIGURE 19. SSH USAGE ON INTERNET SERVERS.....	191
FIGURE 20. LICENSE STREAMS OF A CORE PRODUCT IN A SIMPLIFIED DUAL LICENSING MODEL.	207
TABLE 1. A HISTORICAL TAXONOMY OF THE SOFTWARE INDUSTRY.....	13
TABLE 2. GENERIC SOFTWARE BUSINESS MODELS.....	20
TABLE 3. TYPICAL PROPRIETARY LICENSE RESTRICTIONS.....	26
TABLE 4. SOFTWARE LICENSES AND SERVICES REVENUE OF SOME OF THE WORLD'S LARGEST SOFTWARE PRODUCT COMPANIES	28
TABLE 5. PUBLIC GOODS SUCH AS FREE SOFTWARE ARE NON-EXCLUDABLE AND NON-RIVAL .53	
TABLE 6. USER CAPABILITIES AND TECHNICAL COPY PROTECTION SYSTEMS.	104
TABLE 7. DIFFERENT FORMS OF LEGAL PROTECTION OF SOFTWARE IN THE INDUSTRY HISTORY.	107
TABLE 8. MOST USED OPEN SOURCE LICENSES ON PROJECTS HOSTED AT SOURCEFORGE.	122

TABLE 9. COPYRIGHT FUNCTIONALITY IN DIFFERENT LICENSE TYPES.	162
TABLE 10. COMPARISON OF DIFFERENT IPR DEFENSE OPTIONS FOR OPEN SOURCE DEVELOPERS.	173
TABLE 11. TOTAL NUMBER OF US PATENTS GRANTED FROM 1984 TO EARLY AUGUST 2004 SOURCE: USPTO	176
TABLE 12. MICROCOMPUTER OPERATING SYSTEM COMPETITION IN THE EARLY 2000S.....	194
TABLE 13. MAIN LICENSING OPTIONS IN DIFFERENT LOGICAL LEVELS OF THE OPERATING SYSTEM.	195
TABLE 14. SOME ATTRIBUTES OF THE STUDIED OPEN SOURCE PRODUCTS.. . . .	209

REFERENCES

Note: all links to web pages were checked and in function in early 2005. Years in on-line sources refer to the last modification year of that page when it was referred to. Historical web pages can be searched from Internet Archive, <http://www.archive.org/>

Articles, Books and Reports

Alford, William P. (1995): *To Start a Book is an Elegant Offense: Intellectual Property Law in Chinese Civilization*, Stanford University Press

Allison, John R. and Lemley, Mark A. (1998). "Empirical Analysis of the Validity of Litigated Patents", *American Intellectual Property Law Association Quarterly Journal*, Volume 26, Issue 3, pp. 185-275

Applied Data Research (2003): *Software Products Division Records 1959-1987*, Charles Babbage Institute, University of Minnesota, Minneapolis. Description of the collection is available at <http://www.cbi.umn.edu/collections/inv/cbi00154.html>

Arora, Ashish and Fosfuri, Andrea and Gambardella, Alfonso (2004): *Markets for Technology: The Economics of Innovation and Corporate Strategy*, MIT Press

Arrow, Kenneth J. (1962): "Economic welfare and the allocation of resources for invention", in Richard R. Nelson (ed.): *The Rate and Direction of Inventive Activity*, Princeton University Press, Princeton.

Bakels, Reinier (2002): *The Patentability of Computer Programmes*, European Parliament Directorate-General for Research Working Paper, 2002

Band, Jonathan and Kato, Masanobu (1995) *Interfaces on Trial: Intellectual Property and Interoperability in the Global Software Industry*, Westview Press

Bednarek, Michael and Ineschen, Markus (2004). "Patent Pools as an Alternative to Patent Wars in Emergent Sectors", *Intellectual Property & Technology Law Journal*, Volume 16, Issue 7.

Beresford, Keith (2000). *Patenting Software Under the European Patent Convention*, Sweet & Maxwell.

Berry, David M. (2004). "The contestation of code: A preliminary investigation into the discourse of the free/libre and open source movements", *Critical Discourse Studies*, Volume 1, Issue 1, pp. 65-89.

Besen, Stanley M. (1987): *New Technologies and Intellectual Property: An Economic Analysis*. A Rand Note. Santa Monica, California

- Bessen, Stanley M. and Raskind, Leo J. (1991) "An Introduction to the Law and Economics of Intellectual Property", *Journal of Economic Perspectives*, Volume 5, Issue 1, pp. 3-27.
- Bessen, James and Maskin, Eric (2000). "Sequential Innovation, Patents, and Imitation", working paper.
- Bessen, James and Hunt, Robert M. (2003) "An Empirical Look at Software Patents", National Bureau of Economic Research Working Paper #2003-17
- Bessen, James (2004) "Patent Thickets: Strategic Patenting of Complex Technologies", working paper.
- Bettierle, Richard S. and Davison-Jenkins, Dominic J. (2001). "Coverage concepts for intellectual property", *Risk Management*, Volume 48, Number 2, pp. 17-21.
- Blund, Knut and Edler Jakob and Nack, Ralph and Strauss, Joseph: *Mikro- und makroökonomische Implikationen der Patentierbarkeit von Softwareinnovationen*, Bundesministerium für Wirtschaft und Technologie, 2001. Available at <http://www.bmwi.de/Navigation/Service/bestellservice,did=21760.html>
- Borenstein, Severin and Farrell Joseph and Jaffe, Adam B. (1998): "Inside the Pin-Factory: Empirical Studies Augmented by Manager Interviews", *Journal of Industrial Economics*, Volume 46, Number 2, pp. 123-124.
- Braunstein, Yale M. and Fisher, Dietrich M. and Ordover, Janusz A. and Baumol William J. (1979): "Economics of Property Rights as Applied to Computer Software and Data Bases. Overview of Issues", in George P. Bush and Robert H. Dreyfuss (editors): *Technology and Copyright*. Revised edition. Lomond Books, Mt. Airy, Maryland.
- Bride, Edward (2002): "CBI conference - "Unbundling history: The emergence of the software product"", *IEEE Annals of the History of Computing*, Volume 24, Issue 1, web extras, available at <http://www.computer.org/annals/articles/a1-2002/eands.htm>
- Brooks, Fredenk P. (1975): *The Mythical Man-Month*, Addison-Wesley
- Calabresi, Guido (1970) *The Costs of Accidents: A Legal and Economic Analysis*. Yale University Press
- Campbell-Kelly, Martin (2003): *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*. MIT Press.
- Carr, Nicholas G. (2004) *Does IT Matter? Information Technology and the Corrosion of Competitive Advantage*. Harvard Business School Press.
- Castells, Manuel (1996): *The Rise of the Network Society*. Blackwell Publishers.
- Chávez, Andrea and Tornabene, Catherine and Wiederhold, Gjo (1998) "Software Component Licensing. A Primer", *IEEE Software*, Volume 15, Issue 5, pp. 47-53
- Chesbrough, Henry (2003): *Open Innovation. The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press.

Coase, Ronald H. (1937): "The Nature of the Firm", *Economica*, Vol. 4, No. 16 pp 386-405.

Conner, Kathleen Reavis and Rumelt, Richard P. (1991): "Software Piracy: An Analysis of Protection Strategies", *Management Science*, Volume 37, Number 2, pp 125-139

CONTU (1978). *Final Report of the National Commission on New Technological Uses of Copyrighted Works*, July 31, 1978. Commerce Clearing House, Chicago.

Cusumano, Michael A. (2004) *The Business of Software*. The Free Press

Daffara, Carlo and Carlo González-Barahona, Jesús M. (ed.) (2000): *Free Software / Open Source: Information Society Opportunities for Europe?* Working Group on Libre Software, Version 1.2, available at <http://eu.conecta.it/paper/paper.html>

Dasgupta, Partha David Paul A. (1994). "Toward a new economics of science", *Research Policy*, Volume 23, pp 487-521.

David, Paul A. and Greenstein, Shane (1990): "The Economics of Compatibility Standards. An Introduction to Recent Research", *Economics of Innovation and New Technology*, Volume 1, Issue 1, pp. 3-41

David, Paul A. and Foray, Dominique and Hall, Bronwyn H. and Kahin, Brian and Steinmueller, W. Edward: "Is there really a good economic rationale for an EU Directive on Software Patents?", working paper, 14 July 2003

Davis, Steven J. and MacCracken, Jack and Murphy, Kevin M. (2001): "Economic Perspectives on Software Design: PC Operating Systems and Platforms", *National Bureau of Economic Research Working Paper Series*, paper #8411, available at <http://www.nber.org/papers/w8411>

Dietz, Adolf (1994). "The Artist's Right of Integrity Under Copyright Law – A Comparative Perspective", *IIC*, Volume 25, Number 2, pp 177-194.

Dixit, Avinash K. (2004): *Lawlessness and Economics: Alternative Modes of Governance*. Princeton University Press.

Dosi, Giovanni (1982): "Technological paradigms and technological trajectories", *Research Policy*, Volume 11, pp. 147-162.

Drabos, Peter and Braithwaite, John (2002): *Information Feudalism. Who owns the knowledge economy?* Earthscan

Dravis, Paul (2003). *Open Source Software. Perspectives for Development*. The Dravis Group

Dreier, Thomas (1991): "The Council Directive of 14 May on the Legal Protection of Computer Programs", *European Intellectual Property Review*, Volume 13, Issue 9, pp 319-330

Drucker, Peter (1990). *Managing the Non-Profit Organization: Principles and Practices*. Harper Collins Publishers

- Economides, Nicholas (1996): "The Economics of Networks", *International Journal of Industrial Organization*, Volume 14, Number 6, pp. 673-700.
- Ellickson, Robert C. (1991): *Order without Law*. Harvard University Press.
- Eppler, Dennis and Raviv, Arthur (1978): "Product Safety: Liability Rules, Market Structure, and Imperfect Information", *The American Economic Review*, Volume 68, Issue 1 (Mar., 1978), pp. 80-95.
- Epstein, Richard A. (1997): "Law and Economics: Its Glorious Past and Cloudy Future", *University of Chicago Law Review*, pp. 1167-1174.
- Farrell, Joseph and Monroe, Hunter K. and Saloner, Garth (1998): "The Vertical Organization of Industry: Systems Competition versus Components Competition", *Journal of Economics & Management Strategy*, Volume 7, Issue 2, pp. 143-182.
- Farrell, Joseph and Klemperer, Paul (2001): "Coordination and Lock-In: Competition with Switching Costs and Network Effects", working paper, <http://emlab.berkeley.edu/users/farrell/ftp/lockin.pdf>
- Feller, Joseph and Fitzgerald, Brian (2002), *Understanding Open Source Software Development*, Addison-Wesley.
- Ferhstman, Chaim and Gandal, Neil (2004): "The Determinants of Output Per Contributor in Open Source Projects: An Empirical Examination", working paper, <http://ideas.repec.org/s/cpr/ceprdp.html>
- Fink, Martin (2002). *The Business of Linux and Open Source*. Prentice Hall.
- Foray, Dominique (2004): *Economics of Knowledge*. MIT Press.
- Garzarelli, Giampaolo (2003). "Open Source Software and the Economics of Organization", working paper, <http://opensource.mit.edu/papers/garzarelli.pdf>
- Goldstein, Paul (2001). *International Copyright*, Oxford University Press.
- Gottinger, Hans-Werner (2003). *Economics of Network Industries*. Routledge.
- Grad, Burton (2002): "A Personal Recollection: IBM's Unbundling of Software and Services", *IEEE Annals of the History of Computing*, Volume 24, Issue 1, pp. 64-71.
- Granstrand, Ove (1999). *The Economics and Management of Intellectual Property*. Edward Elgar.
- Grindley, Peter (1995). *Standards, Strategy, and Policy*. Oxford University Press.
- Guadamuz, Andrés (2004): "Viral contracts or unenforceable documents? Contractual validity of copyleft licenses", *European Intellectual Property Review*, Volume 26, Issue 8, pp. 331-339.
- Hahn, Rober W. (ed.) (2002). *Governament Policy Towards Open Source*. AEI Brookings Center.
- Hardin, Garrett (1968): "The Tragedy of the Commons", *Science*, Volume 162, pp. 1243-1248.

Hart, Robert and Holmes, Peter and Reid, John: *The Economic Impact of Patentability of Computer Programs*, Report to the European Commission on behalf of Intellectual Property Institute, London 2000. Available at

http://europa.eu.int/comm/internal_market/en/indprop/comp/studyintro.htm

Hayek, Friedrich A. (1945): "The Use of Knowledge in Society", *American Economic Review*, Volume 35, Issue 4, pp. 519-530

Heller, Michael A. (1998): "The Tragedy of the Anti-Commons: Property in the Transition from Marx to Markets", *Harvard Law Review*, Volume 111, Issue 3, pp. 621-688

Himanen, Pekka (2001): *The Hacker Ethic and the Spirit of the Information Age*. Random House.

von Hippel, Eric (1988): *Sources of Innovation*. Oxford University Press.

von Hippel, Eric (2002): "Open Source Projects as Horizontal Innovation Networks - By and For Users", MIT working paper

Humphrey, Watts S. (2002): "Software unbundling: a personal perspective", *IEEE Annals of the History of Computing*, Volume 24, Issue 1, pp. 59-63.

Jaffe, Adam B. and Lerner, Josh (2004). *Innovation and Its Discontents*. Princeton University Press

Jaeger, Till and Metzger, Axel (2001). *Open Source Software: Rechtliche Rahmenebedingungen der Freien Software*. Verlag C H Beck

Janse, Christian (2003): "Economic Effects of the New German Copyright Contract Law", working paper, available at <http://ideas.repec.org/p/wpa/wuwp/0302003.html>

Johnsen, Torkil C. (1969) "Om patenterbarheden af EDB-programmer" (English Summary: On the Patentability of Computer Software), *Nordiskt Immateriellt Rättsskydd*, Volume 39, Issue 2, pp. 165-168.

Katz, Michael and Shapiro, Carl (1985), "Network Externalities, Competition and Compatibility," *American Economic Review*, Volume 75, Issue 3, pp. 424-440.

Kelly, Kevin (1998). *New Rules for the New Economy*. Viking Press

Kelty, Christopher M. (2001): "Free Software/Free Science", *First Monday*, Volume 6, Issue 12, available at http://www.firstmonday.org/issues/issue6_12/kelty/index.html

Kenney, Martin (2000): "Introduction", in Kenney, Martin (ed.): *Understanding Silicon Valley*, Stanford University Press, pp. 1-12.

Kitch, Edmund W. (1977). "The Nature and Function of the Patent System", *Journal of Law and Economics*, Volume 20, pp. 265-290

Koktvedgaard, Mogens (1968) "Elektronisk databehandling. Immaterialretlige aspekter" (English Summary: Computers and the Law of Intellectual Property), *Nordiskt Immaterialt Rättskydd*, Volume 36, Issue 2, pp. 139-151

Kuhn, Thomas (1962). *The Structure of Scientific Revolutions* University of Chicago Press.

Landes, William M. and Posner, Richard (1989) "An Economic Analysis of Copyright Law", *Journal of Legal Studies*, Volume 18, Issue 2, pp. 325-363.

Landes, William M. and Posner, Richard A. (2003) *The Economic Structure of Intellectual Property Law*, Harvard University Press.

Lanjouw, Jean O. and Pakes, Ariel and Putnam, Jonathan (1998). "How to Count Patents and Value Intellectual Property: The Uses of Patent Renewal and Application Data", *Journal of Industrial Economics*, Volume 46, Issue 4, pp. 405-432

Lanjouw, Jean O. and Schankermann, Mark (2001): "Characteristics of Patent Litigation: A Window on Competition", *RAND Journal of Economics*, Volume 32, Issue 1, pp 129-51

Lanjouw, Jean O. and Schankermann, Mark (2004) "Protecting Intellectual Property Rights: Are Small Firms Handicapped?" *Journal of Law and Economics*, Volume 47, Issue 1, pp. 45-74

Lazonick, William (2003). "Understanding Innovative Enterprise. Towards the Integration of Economic Theory and Business History", in Amatori, Franco and Jones, Geoffrey and Galambos, Louis (eds.): *Business History around the World*, Cambridge University Press, pp. 31-61.

Leibenstein, Harvey (1950): "Bandwagon, snob, and Veblen effects in the theory of consumers demand", *Quarterly Journal of Economics*, Volume 64, Issue 2, pp. 183-207.

Lemley, Mark A. (2001): "Rational Ignorance at the Patent Office", *Northwestern University Law Review*, Volume 95, Issue 4, pp. 1497-1532.

Lemley, Mark A. (2002): "Intellectual Property Rights And Standard-Setting Organizations", *California Law Review*, Volume 89, pp. 1889-1980

Lemley, Mark A. and Shapiro, Carl (2004) "Probabilistic Patents", working paper

Lerner, Josh and Tirole, Jean (2002): "Some Simple Economics of Open Source", *Journal of Industrial Economics*, Volume 52, pp. 197-23

Lerner, Josh and Tirole, Jean (2005), "The Scope of Open Source Licensing", forthcoming in *Journal of Law, Economics and Organization*.

Lessig, Lawrence (2001): *The Future of Ideas Fate of the Commons in a Connected World* Random House

Lessig, Lawrence (2004): *Free Culture. How Big Media Uses Law and Technology to Control Creativity* Penguin Books

Levin, Richard C. and Klevorick, Alvin K. and Nelson, Richard R. and Winter, Sidney (1987). "Appropriating the Returns from Industrial R&D", *Brookings Papers on Economic Activity*, Volume 3, pp. 783-820.

Liebowitz, Stan (1985). "Copying and Indirect Appropriability. Photocopying of Journals", *Journal of Political Economy*, Volume 93, Issue 5, pp. 945-957.

Liebowitz, Stan J. and Margolis, Stephen E. (2001) *Winners, Losers & Microsoft. Competition and Antitrust in High Technology*. Revised edition. The Independent Institute.

Liebowitz, Stan J. (2002) *Re-Thinking the Network Economy*. Amacom.

Locke, John (1690): *Second Treatise on Government*, available at <http://www.swan.ac.uk/poli/texts/locke/lockcont.htm>

Luckombe, Philip (1771). *The History and Art of Printing*. Republished in 1965 by Gregg Press Ltd, London.

Machlup, Fritz (1958) "An Economic Review of the Patent System", Study no. 15 of the Subcommittee on Patents, Trademarks, and Copyrights of the Committee on the Judiciary, United States Senate, 85th Congress, Second Session (Washington, D.C.).

Machlup, Fritz (1962). "The Supply of Inventors and Innovations", in Richard R. Nelson (editor): *The Rate and Direction of Inventive Activity: Economic and Social Factors* Princeton University Press, Princeton

Mattel, Ugo (1997): *Comparative Law and Economics*. The University of Michigan Press

Matthews, Christopher May (2000): *The Global Political Economy of Intellectual Property Rights*, Routledge

McKean, Roland M. (1970): "Products Liability: Implications of Some Changing Property Rights", *The Quarterly Journal of Economics*, Volume 84, Issue 4, pp. 611-626

McKusick, Marshall Kirk (1999): "Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable", in DiBona, Chris et al (eds.) (1999): *Open Sources: Voices from the Open Source Revolution*, O'Reilly, pp. 31-46

Melamed, Douglas and Stoeppelwerth, Ali M. (2002) "The CSU Case: Facts, Formalism And The Intersection Of Antitrust And Intellectual Property Law", *George Mason Law Review*

Menell, Peter S. (1989). "An Analysis of the Scope of Copyright Protection for Application Programs", *Stanford Law Review*, Volume 41, pp. 1045-1104

Merges, Robert (2004): "A New Dynamism in the Public Domain", *University of Chicago Law Review*, pp. 183-203

Messerschmitt, David G. and Szyperski, Clemens (2003). *Software Ecosystem. Understanding an Indispensable Technology and Industry*. MIT Press.

National Research Council (1991). *Intellectual Property Rights Issues in Software*. National Academy Press.

Negroponte, Nicholas (1995) *Being Digital*. Knopf

Newcity, Michael A. (1978): *Copyright Law in the Soviet Union*. Praeger Publishers, New York

Nichols, Kenneth (1996) *Inventing Software*, Quorum Books

North, Douglass C. (1981): *Structure and Change in Economic History*. W.W. Norton & Company

Novos, Ian E. and Waldman, Michael (1986): "The Emergence of Copying Technologies: What Have We Learned?", *UCLA working paper*, <http://www.econ.ucla.edu/workingpapers/wp408.pdf>

OECD (2002) *Report of the OECD Task Force on Software Measurement in the National Accounts*. Organisation for Economic Co-operation and Development, Paris.

Oksanen, Ville and Valmaki, Mikko (2002): "Transnational Advocacy Network Opposing DRM - Technical and Legal Challenge to Media Companies", *Journal of Media Management*, Volume 4, Issue 3.

O'Mahony, Siobhan (2003). "Guarding the Commons. How Community Managed Software Projects Protect Their Work", *Research Policy*, Volume 32, Issue 7, pp. 1179-1198

Osorio, Carlos A. (2002): "A Contribution to the Understanding of Illegal Copying of Software", *MIT Working paper*, available at <http://opensource.mit.edu/>

Palmer, Tom G. (1989): "Intellectual Property. A Non-Posnerian Law and Economics Approach", *Harvard Law Review*, <http://www.tomgpalmer.com/>.

Patterson, Lyman Ray (1968). *Copyright in Historical Perspective*, Vanderbilt University Press

PbT Consultants (2001): *The Results of the European Commission Consultation Exercise on the Patentability of Computer Implemented Inventions*, 2001

Posner, Richard (2002) *Antitrust Law*. 2nd edition, The University of Chicago Press.

Prasada, Ashutosh and Mahajan, Vijay (2003) "How many pirates should a software firm tolerate?", *International Journal of Research in Marketing*, Volume 20, Issue 4, pp. 337-353

Puckett, Allen W. (1966): "The Limits of Copyright and Patent Protection for Computer Programs", in *Copyright Law Symposium Number Sixteen: Nathan Burkan Memorial Competition Sponsored by the American Society of Composers*. Columbia University Press, New York 1968, pp. 81-142.

Pugh, Emerson W. (2002) "Origins of Software Unbundling", *IEEE Annals of the History of Computing*, Volume 24, Issue 1, pp. 59-63

- Rahnasto, Ilkka (2003) *Intellectual Property Rights, External Effects, and Anti-trust Law Leveraging IPRs in the Communications Industry* Oxford University Press
- Rajala, Risto and Rossi, Matti and Tuunainen, Virpi Kristina and Korri, Santen (2001). *Software Business Models. A Framework for Analysing Software Industry*. Technology Review 108/2001. Finnish National Technology Agency. <http://www.tekes.fi>
- Raymond, Eric S. (2001). *The Cathedral and the Bazaar*, 2nd Edition, O'Reilly
- Rehn, Alf (2001) *Electronic Potlatch – a study on new technologies and primitive economic behaviors*, Royal Institute of Technology, Stockholm
- Riss, Thomas (1996). *Ophavsret og Retsekonomi. Immaterielle goder i kulturøkonomisk betydsning*. (With English summary) Gadjura
- Rose, Carol M (1986). "The Comedy of the Commons: Custom, Commerce, and Inherently Public Property", *University of Chicago Law Review*, Volume 53, Issue 3, pp 711-781.
- Rosen, Lawrence E. (2004): *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall.
- Rushton, Michael (1998). "The Moral Rights of Artists: Droit Moral ou Droit Pécuniaire?", *Journal of Cultural Economics*, pp. 15-32
- Samelson, K. and Bauer P. L. (1960) "Sequential formula translation", *Communications of the ACM*, Volume 3, Issue 2, pp. 76-83
- Samuelson, Pamela and Davis, Randall and Kapor, Mitchell D. and Reichman J.H. (1994): "A Manifesto Concerning the Legal Protection of Computer Programs", *Columbia Law Review*, Volume 94, Issue 8, pp. 2308-2431.
- Sarath, Bharat (1991) "Uncertain Litigation and Liability Insurance", *The RAND Journal of Economics*, Volume 22, Issue 2, pp. 218-231
- Schumpeter, Joseph (1942): *Capitalism, Socialism and Democracy* Harper & Brothers.
- Shavell, Steven (1982): "On Liability and Insurance", *The Bell Journal of Economics*, Volume 13, Issue 1, pp. 120-132.
- Shapiro, Carl (1989): "The Theory of Business Strategy", *The RAND Journal of Economics*, Volume 20, Issue 1., pp. 125-137
- Shapiro, Carl and Varian, Hal R. (1998) *Information Rules: A Strategic Guide to the Network Economy* Harvard Business School Press
- Shapiro, Carl (2001): "Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard Setting," in Jaffe, Adam and Lerner, Joshua and Stern, Scott (eds): *Innovation Policy and the Economy*, MIT Press.
- St Laurent, Andrew (2004): *Understanding Open Source & Free Software Licensing*. O'Reilly.

- Stallman, Richard (2002) *Free Software, Free Society: Selected Essays by Richard M. Stallman*, O'Reilly.
- Stalle, Alexander (2002) *Future of the Past* Picador
- Shy, Oz and Thussie, Jacques (1999), "A Strategic Approach to Software Protection", *Journal of Economics and Management Strategy* Volume 8, Issue 2, pp. 163-90
- Shy, Oz (2001) *The Economics of Network Industries*. Cambridge University Press.
- Takeyama, Lisa (1994): "Distributing Experience Goods by Giving Them Away. Shareware – Some Stylized Facts and Estimates of Revenue and Profitability," *Economic Innovation and New Technology*, Volume 3, Issue 2, pp. 161-174
- Takeyama, Lisa (1994b) "The Welfare Implications of Unauthorized Reproduction of Intellectual Property in the Presence of Demand Network Externalities", *The Journal of Industrial Economics*, Volume 42, Issue 2, pp. 155-166
- Teece, David J. (1986) "Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy", *Research Policy*, Volume 15, Issue 6, pp. 285-305.
- Teece, David J. (2000) *Managing Intellectual Capital*. Oxford University Press.
- Torrisi, Salvatore (1998). *Industrial Organisation and Innovation. An International Study of the Software Industry* Edward Elgar
- Torvalds, Linus, and Diamond, David (2001). *Just for Fun: The Story of an Accidental Revolutionary*. Harper Business
- Tuoma, Ilkka (2002) *Networks of Innovation* Oxford University Press.
- Tyler, Michael A. (1986) "The Extent of Software Piracy", in Hubard, Frank L. and Shelton, R. D. (eds.) (1986) *Protection of Computer Software*. Law & Business Inc
- Viscusi, W. Kip and Moore, Michael J. (1993) "Product Liability, Research and Development, and Innovation", *The Journal of Political Economy*, Volume 101, Issue 1, pp. 161-184.
- Valmaki, Mikko (2003a): "Dual Licensing in Open Source Software Industry", *Systèmes d'Information et Management*, Volume 8, Issue 1, pp. 63-75.
- Valmaki, Mikko (2003b): "From individuals to political institutions - The discourse on institutional change in free software and open source communities", *Medium*, Volume 2, Issue 1, available at <http://www.m-cult.net/medium/>.
- Valmaki, Mikko (2004a): "Ajatuksia tekijänoikeuslainsaadannon uudistamiseksi" (In Finnish, with English summary titled "Some Thoughts on Copyright Law Reform"), *Läksimies*, Volume 102, Issue 2, pp. 255-273.
- Valmaki, Mikko (2004b). "A Practical Approach to the Problem of Open Source and Software Patents", *European Intellectual Property Review*, Volume 26, Issue 12.

Valmaki, Mikko and Hietanen Herkko (2004): "The Challenges of Creative Commons Licensing", *Computer Law Review*, Volume 5, Issue 6.

Välimäki, Mikko and Oksanen, Ville (2005): "The Impact of Free and Open Source Licensing on Operating System Software Markets", *Telematics and Informatics*, Volume 22, Issues 1-2, pp. 97-110

Watt, Richard (2000): *Copyright and Economic Theory*, Edward Elgar.

Williams, Sam (2002): *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly

West, Joel and Dedrick, Jason (2001): "Open Source Standardization: The Rise of Linux in the Network Era," *Knowledge, Technology & Policy*, Volume 14, Issue 2, pp. 88-112

West, Joel (2003): "How open is open enough? Melding proprietary and open source platform strategies", *Research Policy*, Volume 32, pp. 1259-1285.

von Westrap, Falk (2003): *Modeling Software Markets. Empirical Analysis, Network Simulations and Marketing Implications*. Physica-Verlag

News, Interviews and Online-sources

Akerlof et al. (2002) *Amici Curiae, Eldred v. Ashcroft*, May 20th, 2002, available at <http://econ.law.harvard.edu/openlaw/eldredvashcroft/supct/amici/economists.pdf>.

Anderson, Ross (2003) "Trusted Computing" Frequently Asked Questions - TC / TCG / LaGrande / NGSCB / Loughorn / Palladium / TCPA, Version 1.1, August 2003, <http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html>

Apache (2004): "Apache License v2.0 and GPL Compatibility", available at <http://www.apache.org/licenses/GPL-compatibility.html>

Apple (2003) Darwin FAQ, available at <http://developer.apple.com/darwin/projects/darwin/faq.html>

Asami, Naoki (2001): "30th Anniversary Interview of Linus Torvalds", Nikkei Electronics Online, available at <http://ne.nikkeibp.co.jp/english/2001/30ann/int2>

Associated Press (1994) "Microsoft Loses Patent Suit", Associated Press, February 23, 1994

Barlow, John Perry (1994) "The Economy of Ideas", *Wired* 2.03

Barr, Joe (2004): "HP memo forecasts MS patent attacks on free software", NewsForge, 19th July 2004.

Bigelow, Robert P (1970): "Contract Caveats", *Defamation*, 1970, Volume 16, Number 11, pp. 41-44

Boyle, James (2004): "Give me liberty and give me death?", *Financial Times*, October 21, 2004.

Bowman, Lisa M. (2002). "Open-source Visionary. Proprietary software is not okay", ZD Net AU, 10th December 2002, available at <http://www.zdnet.com.au/news/tech/os/story/0,2000024997,20270554,00.htm>

Broersma, Matthew (2002). "Eric Raymond: Why open source will rule", ZD Net News, 29th March 2002, available at <http://zdnet.com.com/2100-1104-871366.html>

Brown, Glenn Obs (2004) "Announcing (and explaining) our new 2.0 licenses", 25th May, 2004, <http://creativecommons.org/weblog/entry/4216>

Business Software Alliance (2004): *First Annual BSA and IDC Global Software Piracy Study*, available at <http://www.bsa.org/>

CJA Consultants (2003): *Patent Litigation Insurance. A Study for the European Commission on possible insurance schemes against patent litigation risks*. Final Report.

Codewalkers (2002), Interview of Michael Widenius, http://codewalkers.com/interviews/Monty_Widenius.html

Cohen, Nancy (2003). "What's GNU? GPL v3 and ASPs ", *Open Magazine*, http://www.open-mag.com/features/Vol_66/GNU/GNU.htm

Computer Business Review (2005): "CA confirms plans for open source patent pledge", 3 March, 2005, available at http://www.cbronline.com/article_news.asp?guid=DC531E86-85D0-42C4-A752-DF45DA446142

Computer Hope (2003): "History of Microsoft Windows", available at <http://www.computerhope.com/history/windows.htm>

ComputerWise (2002): "Clock ticks on Microsoft's licensing - opposition remains ", *The Register*, July 31, 2002 http://www.theregister.co.uk/2002/07/31/clock_ticks_on_microsofts_licensing/

Conner, Doug (1998): "Father of DOS still having fun at Microsoft", *Microsoft Micronews*, April 10, 1998. Available at http://www.patersonstech.com/Dos/Micronews/paterson04_10_98.htm

The Economist (2001): "Software Survey", *The Economist*, 21 April, 2001.

The Economist (2004a): "Sir Bill and his dragons—past, present and future", *The Economist*, 29 January, 2004

The Economist (2004b): "BRAIN SCAN. Unix's founding fathers ", *The Economist*, 10 June, 2004.

European Information Technology Observatory (2004). "Out of the tunnel: Western European and worldwide markets for Information Technology and Telecommunications (ICT) are picking up", press release, 16th February 2004

Farber, Dan (2003): "Unplugged: Mårten Mickos, CEO MySQL AB", *ZDNet Tech Update*

Fichera, Richard (2004): "Linux IP Litigation: Users Largely Unconcerned About SCO Suit, Indifferent To Indemnification", *Forrester Research*, 14th May 2004, summary available at <http://www.forrester.com>

Ford, Nelson (2000): "The History of Shareware & PsL", available at <http://www.asp-shareware.org/users/history-of-shareware.asp>

Foundation for Free Information Infrastructure (2004): "The TRIPs Treaty and Software Patents", <http://swpat.ffii.org/analysis/trips/>

Free Software Foundation (2002): "Free Software Foundation Announces Support of the Affero General Public License, the First Copyleft License for Web Services", <http://www.gnu.org/press/2002-03-19-Affero.html>

Free Software Foundation (2003): "BSD License Problem", <http://www.gnu.org/philosophy/bsd.html>

Free Software Foundation (2004): "Frequently Asked Questions about the GNU GPL", available at <http://www.gnu.org/licenses/gpl-faq.html>

Free Software Foundation (2004b): "Various Licenses and Comments about Them", <http://www.gnu.org/licenses/license-list.html>

Fromy, Philippe (2001): "Interview: Trolltech's President Erik Eng", <http://dot.kde.org/1001294012/>

Fried, Iva (2004): "Gates wants patent power", *News.com*, 29th July 2004

Gates, William H. (1976): "An Open Letter to Hobbyists", 3rd February 1976, available at <http://www.blinkenlights.com/classicomp/gateswhine.html>

Geber, Jason (2004): "Government Open Source Policies", http://www.csis.org/tech/OpenSource/0408_ospolicies.pdf

Google Zeitgeist (2004): <http://www.google.com/press/zeitgeist/archive.html>

Greene, Thomas C. (2001): "Ballmer: 'Linux is a cancer'", *The Register*, June 2nd 2001, http://www.theregister.co.uk/2001/06/02/ballmer_linux_is_a_cancer/

Hewlett-Packard (2003): "HP indemnifies Linux customers against SCO lawsuit", 24th September 2003, available at <http://www.hp.com/>

Hirsch, Phil (1966): "The Patent Office Examines Software", *Datamation*, November 1966, pp. 79-81

Hirsch, Phil (1968): "CCPA Reconsiders Patent Decision and Prater & Wei Wait 'Er and Pray'", *Datamation*, April 1968, pp. 174-175

Hirsch, Phil (1969): "Conference Considers Software Legal Issues and It's Good News Two to One", *Datamation*, November 1969, pp. 357-359.

- Howard, James (2001): "The BSD Family Tree", *Daemon News*, available at http://www.daemonnews.org/200104/bsd_family.html
- Hubbard, Jordan (2003): "A Brief History of FreeBSD", available at http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/history.html
- Greant, Zak (2002): The Future of MySQL, presentation at SD Expo 24-4 2002, San Jose, CA, USA, <http://www.mysql.com/information/presentations/index.html>
- IBM (2002): Common Public License Frequently Asked Questions, <http://www-106.ibm.com/developerworks/library/os-cplfaq.html>, June 1, 2002
- IBM (2005): "IBM Statement of Non-Assertion of Named Patents Against OSS", available at <http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf>
- IDC (2003): "IDC Says Microsoft Is Moving into Dominant Role in Server Operating Environments, Even as Linux Grows", IDC press release, 8th October 2003.
- Initiative for Software Choice (2004): "Initiative for Software Choice – Key Messages", http://www.softwarechoice.org/download_files/Key_JSC_messaging.pdf
- InnoDB (2004): "MySQL/InnoDB (= MySQL Pro) commercial licensing", <http://www.innodb.com/licenses.php>
- Irlam, Gordon (1998): "Software Patents", <http://www.base.com/software-patents/software-patents.html>
- Jo Foley, Mary (2004): "FlexWiki: Microsoft's Third Open Software Project", *eWeek*, September 28, 2004, available at <http://www.eweek.com/article2/0,1759,1657278,00.asp>
- Kane, Margaret (2002): "W3C bows to royalty-free pressure", *News.com*, November 14th, 2002
- Kelly, J. S. (2000): "An interview with Richard Stallman", *LinuxWorld.com*, 29th March, 2000, <http://www.linuxworld.com/Mar/2687/LWD000329rms/>
- Kilgard, Ron et al (1995): *Amicus curiae, Lotus v. Borland*. December 1995. Available at <http://www-swiss.ai.mit.edu/6805/articles/unt-prop/lotus/law-prof-amicus.txt>
- Krim, Jonathan (2003): "The Quiet War Over Open-Source", *Washington Post*, August 21, 2003
- LaMonica, Martin (2004): "Pandora's box for open source", *CNET News.com*, February 12, 2004, available at <http://zdnet.com.com/2100-1104-5157874.html>
- Lawson, Stephen (2004): "Microsoft goes open source with WIX tool", *InfoWorld*, April 5, 2004, http://www.infoworld.com/article/04/04/05/HNwixtool_1.html
- The League for Programming Freedom (1991): *Against Software Patents*, February 28, 1991, available at <http://lpf.mit.edu/Patents/against-software-patents.html>

Leonard, Andrew (2000a): "BSD Unix: Power to the people, from the code", *Salon.com* May 16, 2000, available at http://dir.salon.com/tech/isp/2000/05/16/chapter_2_part_one/index.html?pn=1

Leonard, Andrew (2000b): "License to be good", *Salon.com* September 22, 2000, available at <http://dir.salon.com/tech/col/leon/2000/09/22/licenses/index.html?pn=1>

Livingston, Brad (2000): "Is Microsoft's change in Kerberos security a form of 'embrace, extend, extinguish'?", *InfoWorld*, available at <http://archive.infoworld.com/articles/op/xml/00/05/15/000515oplivingston.xml>

Majerus, Laura A. (2003): "Court Evaluates Meaning of 'Derivative Work' in an Open Source License", *FindLaw.com*.

Malcolm, Jeremy (2003): "Problems in Open Source Licensing", a paper presented at Australia's national Linux conference, January 24, 2003, available at <http://www.law.com.au/public/licencearticle.html>

Markham, Gervase (2004): "Relicensing help wanted", July 14, 2004, <http://weblogs.mozillazine.org/gerv/archives/005992.html>

McCue, Andy (2004): "Gartner: Re-negotiate software license deals now", *News.com*, November 23, 2004

Metzger, Axel (2004): "Free Content Licenses under German Law", talk given at the Wissenschaftskolleg, Berlin, June 17, 2004, available at <http://lists.ibiblio.org/pipermail/cc-de/2004-July/000015.html>

Microsoft (2003a): Microsoft Shared Source Initiative Frequently Asked Questions, available at <http://www.microsoft.com/resources/sharedsource/Initiative/FAQ.msp#x>

Microsoft (2003b): Shared Source Licensing Programs, available at <http://www.microsoft.com/resources/sharedsource/Licensing/default.msp#x>

Microsoft (2003c): Software Assurance, available at <http://www.microsoft.com/licensing/programs/sa/>

Moglen, Eben (2001a): Free Software Matters, Enforcing the GPL I, *Linux User*, available at <http://emoglen.law.columbia.edu/publications/lu-12.html>

Moglen, Eben (2001b): Free Software Matters, Enforcing the GPL II, *Linux User*, available at <http://emoglen.law.columbia.edu/publications/lu-13.html>

Morrison & Foerster (2000): "Diplomatic Conference Votes to Maintain Status Quo Regarding Software Patents in Europe Pending Issuance of a New Software Patent Directive by the European Union", *Legal Update*, November 2000, available at <http://www.mofo.com/news/updates/files/update152.html>

Mozilla Relicensing FAQ (2004): <http://www.mozilla.org/MPL/relicensing-faq.html>

- Mundie, Craig (2001): "The Commercial Software Model and Sustainable Innovation", May 16, 2001, available at http://www.microsoft.com/resources/sharedsource/Initiative/speeches/mundie_model.msp
- MySQL (2001): "FAQ on MySQL vs. NuSphere Dispute", 13 7 2001 <http://www.mysql.com/news/article-75.html>
- MySQL (2004a): "MySQL Licensing Policy", <http://www.mysql.com/company/legal/licensing/>
- MySQL (2004b): "FLOSS License Exception v0.2", <http://www.mysql.com/company/legal/licensing/floss-exception.html>, 15 July 2004.
- Nature (2001): "Future e-access to the primary literature", *Nature*, web debate, <http://www.nature.com/nature/debates/e-access/>
- Netcraft (2004): Web Server Survey, available at http://news.netcraft.com/archives/web_server_survey.html
- Netscape (1998): "Netscape announces Mozilla.org, a dedicated team and web site supporting development of free client source code", press release, February 23, 1998
- Novell (2004): "Novell Supports Enterprise Linux Customers With New Linux Indemnification Program", 13th January 2004, available at <http://www.novell.com/>
- OneStat (2003a): "Search Engine Ratings", press release, May 12, 2003, http://www.onestat.com/html/aboutus_pressbox21.html
- http://www.onestat.com/html/aboutus_pressbox23.html
- OneStat (2003b): "Microsoft's Windows dominates the OS market on the web according to OneStat.com", press release, September 24, 2003, http://www.onestat.com/html/aboutus_pressbox24.html
- OneStat (2004): "Microsoft's Internet Explorer global usage share is 93.9 percent according to OneStat.com", press release, May 28, 2004, http://www.onestat.com/html/aboutus_pressbox30.html
- Open Source Development Labs (2003): "Linux Legal Defense Fund FAQ" Available at <http://www.osdl.org/>
- Open Source Development Labs (2004): "OSDL to Support Enhancements to Linux Kernel Development Process", 24th May 2004. Available at <http://www.osdl.org/>
- Open Source Initiative (1999): "History of the OSI", <http://www.opensource.org/docs/history.php>
- Open Source Risk Management (2004a): "OSRM Certifies Linux Kernel Free of Copyright Infringement" 19th April 2004. Available at <http://www.osriskmanagement.com/>

- Open Source Risk Management (2004b): "Results of First-Ever Linux Patent Review Announced", 2nd August 2004. Available at <http://www.osriskmanagement.com/>
- OpenSSH (2001). "SSH COM Trademark Dispute", <http://www.openssh.org/ssh-dispute/>
- OpenSSH (2004) "Project History and Credits", <http://www.openssh.com/history.html>
- Perens, Bruce (2002) "MS 'Software Choice' scheme a clever fraud", *The Register*, 9th August, 2002, http://www.theregister.co.uk/2002/08/09/ms_software_choice_scheme/
- Pournelle, Jerry (1987) "Back to Work!", *BYTE*, Volume 12, No 4.
- Pournelle, Jerry (1988): "Computing at Chaos Manor Life after Las Vegas", *BYTE*, Volume 13, No 2.
- Proffitt, Brian (2004) "XFree86 License Causes Distros to Rethink Plans", *LinuxToday*, February 18, 2004, <http://linuxtoday.com/developer/2004021803026NWDTL>
- Ravicher, Dan (2002). "Software Derivative Work: A Circuit Dependent Determination", available at http://www.pbwt.com/Attorney/iles/ravicher_1.pdf
- Raymond, Eric S. (2002). "Geeks with Guns!", available at <http://www.catb.org/~esr/geeks-with-guns/>
- Raymond, Eric S. and Landley Rob (2003). "OSI Position Paper on the SCO-vs.-IBM Complaint", available at <http://www.opensource.org/sco-vs-ibm.html>, accessed 20 September 2003
- Red Hat (2004a): "Red Hat Announces Open Source Assurance to Safeguard Customer Investment", 20th January 2004, available at <http://www.redhat.com/>
- Red Hat (2004b). "Red Hat, Inc. Statement of Position and Our Promise on Software Patents", available at http://www.redhat.com/legal/patent_policy.html
- Roblino (2004): "Wikipedia Founder Jimmy Wales Responds", *Slashdot*, 28 July 2004. Available at <http://www.slashdot.org/>
- Rose, Dan (2004) "DOS® Abandonware Utilities -- Disk Copy-Protection Removal Software", <http://home.pmt.org/~drose/aw-dos-37.html>
- Rosen, Lawrence (2001). "Naming Open-Source Software", *Linux Journal*, October 1, 2001
- Rosen, Lawrence (2002) "Allocation of the Risks", *Linux Journal*, September 1, 2002
- Rosen, Lawrence (2004): "Patents in an open source world", *NewsForge*, July 27, 2004
- Salus, Peter H. (1994) "The history of Unix is as much about collaboration as it is about technology", *BYTE com*, November 1994.

Samuelson, Pamela et al (1995): *Amicus curiae, Lotus v. Borland* December 1995 Available at <http://www-swiss.ai.mit.edu/6805/articles/int-prop/lotus/law-prol-amicus.txt>

ScanSSH (2004). "SSH usage profiling", <http://www.openssh.org/usage/>

Schneier, Bruce (2001): "The Futility of Digital Copy Prevention", *Crypto-Gram Newsletter*, May 15, 2001 <http://www.schneier.com/crypto-gram-0105.html>

Shankland, Stephen (2004a): "MySQL addresses open-source license problem", *News.com*, March 12, 2004

Shankland, Stephen (2004b): "IBM pledges no patent attacks against Linux", *News.com*, 4th August 2004.

Singer, Michael (2004) "Apple Sees a Shift in Developer Profiles", *InternetNews*, available at <http://www.internetnews.com/ent-news/article.php/3333591>

Sleepycat (2004): "Sleepycat Software Product Licensing", <http://www.sleepycat.com/download/licensinginfo.shtml>

Stallman, Richard (1986). Lecture at Kungliga Tekniska Hogskolan (Royal Institute of Technology), Stockholm, Sweden, 30th October 1986, available at <http://www.gnu.org/philosophy/stallman-kth.html>

Stallman, Richard (1999a): "Why you shouldn't use the Library GPL for your next library", <http://www.gnu.org/philosophy/why-not-lgpl.html>

Stallman, Richard (1999b): "Saving Europe from Software Patents", *Linux Today*, available at http://features.linuxtoday.com/news_story.php3?tsn=1999-05-16-003-05-NW-LF

Stapleton, Lisa (2004): "Stallman: Accusatory Report Deliberately Confuses", *Linux Insider* 27th May 2004, available at <http://www.linuxinsider.com/story/34069.html>

Stone, Brad (2004): "Nickels, Dimes, Billions", *Newsweek*, web exclusive, 2nd August 2004

SUN (2005): "Sun Grants Global Open Source Community Access to More than 1,600 Patents", January 25, 2005, available at <http://www.sun.com/snu/Press/sunflash/2005-01/sunflash.20050125.2.html>

Tan, Andy (2001): "The History of the GPL", available at http://www.free-software.org/gpl_history/

Talk, Chuck (2004): "An Interview with Matt Asay of Novell", May 31, 2004, <http://rootprompt.org/article.php3?article=6940>

Tomboy (1998): "Reinhalzung Lotus 1-2-3 DOS versions 2.01, 2.3, and 3.1", <http://fravia.anticrack.de/123dos.htm>

Torvalds, Linux and Cox, Alan (2003): An open letter to the European Parliament, available at http://www.elfi.org/patentit/patents_torvalds_cox.txt

TrollTech (2003) "Licenses for Code Used in Qt"

<http://doc.trolltech.com/3.1/licenses.html>

TrollTech (2004). "Qt Open Source Edition Licensing",

<http://www.trolltech.com/products/qt/opensource.html>

US vs Microsoft (2003) United States v Microsoft – Current Case, available at

http://www.usdoj.gov/aic/cases/ms_index.htm

Wheeler, David A. (2004) "Make Your Open Source Software GPL-Compatible Or Else", <http://www.dwheeler.com/essays/gpl-compatible.html>

Zawinski, Jamie (2003) "Emacs Timeline", available at

<http://www.jwz.org/doc/emacs-timeline.html>

Zimran, Ahmed (2001), "Interview with Sleepycat President and CEO Michael Ol-

son", <http://www.winterspeak.com/columns/102901.html>, 29.10.2001

Court Cases, Official Documents and Licenses

Cases – United States

Computer Associates International v Altai (US Court of Appeals, 2nd Circuit, June 22, 1992)

Diamond v Diehr (US Supreme Court, March 3, 1981)

Gottschalk v Benson (US Supreme Court, November 20, 1972)

Litchfield v. Spielberg (US Court of Appeals, 9th Circuit, July 6, 1984)

Lotus Development Corp v Borland International Inc (US Court of Appeals, 1st Circuit, March 9, 1995)

In Re Alappat (US Court of Appeals, Federal Circuit, July 29, 1994)

Parker v. Flook (US Supreme Court, June 22, 1978)

Progress Software Corp v MySQL AB (filed in US District Court for the District of Massachusetts, 2002 – settled in November 7, 2002).

SCO Group v. International Business Machines, Inc (filed in US District Court for the District of Utah, 2003 – ongoing and extensively covered at e.g. <http://www.groklaw.net/>)

Stac Electronics v Microsoft Corp. (filed in US District Court for the Central District of California, 1993 – settled in June 1994).

United States v. Manzer (US Court of Appeals, 8th Circuit, October 27, 1995)

United States v. Microsoft (filed in US District Court for the District of Columbia, 1999 – case settled in November 1, 2002)

Unix System Laboratories, Inc vs Berkeley Software Design, Inc. (filed in US District Court for the District of New Jersey, 1992 – settled in February 4, 1994).

Vault Corp. v. Quaid Software Ltd. (US Court of Appeals, 5th Circuit, June 22, 1988)

Whelan Associates Inc v. Jaslow Dental Laboratory, Inc., et al (US Court of Appeals, 3rd Circuit, August 4, 1986)

Cases – Europe

European Patent Office, Board of Appeal, T 0208/84, VICOM, July 15, 1986

European Patent Office, Board of Appeal, T 1173/97, IBM, July 1, 1998

Landgericht München I, May 19, 2004. Available at
http://www.lbb.de/urteil_lg_muenchen_gpl.pdf

Finnish Supreme Court, KKO 1998/91, August 21, 1998

Finnish Supreme Court, KKO 2003/88, November 10, 2003

Official documents

Berne Convention for the Protection of Literary and Artistic Works, Paris Act of July 24, 1971, as amended on September 28, 1979 (Berne Convention)

Conference of the Contracting States to Revise the 1973 European Patent Convention (Munich, 20 to 29 November 2000) (EPC 2000)

Convention on the Grant of European Patents of 5 October 1973 (European Patent Convention)

Council Directive 91/250/EEC of 14 May 1991 on the Legal Protection of Computer Programs (Software Copyright Directive)

Council Directive 93/13/EEC of 5 April 1993 on unfair terms in consumer contracts (Consumer Contract Directive)

Directive 99/44/EC of the European Parliament and of the Council of 25 May 1999 on certain aspects of the sale of consumer goods and associated guarantees (Consumer Goods and Guarantees Directive)

Examination Guidelines for Computer-Related Inventions, February 28, 1996 (US Software Patent Guidelines)

Guidelines for Examination in the European Patent Office Effective as 1 June 1978

Guidelines for Examination in the European Patent Office December 2003 edition

Memorandum of the Constitutional Law Committee of the Finnish Parliament 7/2005, 10 March, 2005

Munich Diplomatic Conference for the Setting Up of a European System for the Grant of Patents (Munich, September 10 to October 5, 1973).

Promoting Innovation Through Patents - Green Paper on the Community patent and the patent system in Europe, COM(97) 314, June 1997

Proposal for a directive of the European Parliament and of the Council on the patentability of computer-implemented inventions, Brussels, 20.02 2002, COM(2002) 92 final (Software Patent Directive Proposal)

WIPO Advisory Group of Governmental Experts on the Protection of Computer Programs, Geneva, March 8-12, 1971 *Copyright*, Volume 7, Issue 3, pp. 35-40

WIPO Model provisions on the protection of computer software, Geneva, 1978. (WIPO *Sui Generis* Proposal)

WIPO Group of Experts on the Legal Protection of Computer Software (Geneva, June 13 to 17, 1983), *Copyright*, Volume 19, Issue 9, 1983, pp. 271-279

WIPO Group of Experts on the Copyright Aspects of the Protection of Computer Software, Geneva, February 25 to March 1, 1985. *Copyright*, Volume 21, Issue 4, pp. 146-149

WIPO Proposal by Argentina and Brazil for the Establishment of a Development Agenda for WIPO, Geneva, August 24, 2004 (WIPO Development Agenda Proposal)

WTO Treaty on Trade Related Aspects of Intellectual Property Rights, Morocco, April 15, 1994 (TRIPS)

Definitions and policies

Debian Social Contract 1.1 (http://www.debian.org/social_contract.html)

The Free Software Definition (<http://www.gnu.org/philosophy/free-sw.html>)

Open Source Definition (<http://www.opensource.org/docs/definition.php>)

W3C Patent Policy (<http://www.w3.org/Consortium/Patent-Policy/>)

Licenses (all except the Unix License can be found at <http://www.opensource.org/>)

Academic Free License

Apache License 2.0

Artistic license

BSD license

Common Public License

GNU General Public License (GPL)

GNU Lesser General Public License (LGPL)

MIT license

Mozilla Public License 1.1 (MPL)

Open Software License

Qt Public License (QPL)

Sleepycat License

Unix License (1974): "Software Agreement between Western Electric Company, Incorporated and Katholieke Universiteit effective as of 1st December 1974" Available at <http://cm.bell-labs.com/cm/cs/who/dmr/licenses/6thEdlicense.pdf>

INDEX

- Adobe [95](#)
- Academic Free License [121](#)
- Advanced Micro Devices (AMD) [104](#)
- Affero Public License [138](#)
- Amazon [138](#)
- Amdahl [90](#)
- Amstrad [89](#)
- antitrust law *see* competition law
- Apache [2](#), [17](#), [21](#), [37](#), [59](#), [121](#), [140](#), [165](#)
License [121](#), [152-153](#), [180](#)
- Apple [2](#), [37](#), [39](#), [89-90](#), [121](#), [165](#), [176](#), [194](#)
 Darwin [201-202](#)
 Mac OS [19](#), [192-194](#), [200-202](#), [204-205](#)
 Apple Public Source License [121](#), [195](#), [201](#)
- Application Service Providing (ASP) [137](#)
- Applied Data Research (ADR) [82-83](#)
- Artistic License [120-121](#), [154](#)
- Atari [192](#)
- AT&T [30-32](#), [35](#), [90](#), [192](#)
- Autodesk [95](#)
- Axmark, David [211](#)
- Ballmer, Steve [131](#)
- Band, Jonathan [88](#)
- Bakels, Resmer [184](#)
- Berkeley Software Design [32](#)
- Berkeley Software Distribution (BSD) [30-32](#), [35-36](#), [193](#), [200-202](#)
 FreeBSD [33](#), [138](#), [200](#)
 OpenBSD [33](#), [190](#), [200](#)
 NetBSD [33](#), [200](#)
 BSD License [33](#), [151-152](#), [181](#), [195](#), [201](#), [209](#)
- Berkeley Softworks [196](#)
 GeoWorks [196](#)
- Besen, Stanley [62](#)
- Blind, Knuth [184](#)
- Borland [90-91](#)
- Braunstein, Yale M. [64](#)
- Broderbund Software [90](#)
- Bull [89](#)
- business history
 continuing approach [6-7](#)
- Business Software Alliance (BSA) [61](#), [110](#)
- BSD *see* Berkeley Software Distribution
- Calabresi, Guido [166-179](#)
- Caldera [35](#)
- Campbell-Kelly, Martin [13](#), [23](#)
- Chandler, Alfred D. [6](#)
- Chesbrough, Henry W. [76](#)
- Christensen, Clayton M. [6](#)
- closed source *see* proprietary
- Coase, Ronald H. [75](#)
- Commodore [192](#)
- Common Public License (CPL) [121](#), [142-143](#), [150](#), [180](#)
- community norms *see* social norms
- Compaq [90](#)
- compatibility
 definition of [54-56](#)
 GPL-compatibility [140](#)
 and fragmentation [55-56](#)
 between licenses [115](#), [123](#), [159](#), [162-163](#)
- competition law
 IBM antitrust investigation [22](#)
 and intellectual property [78-80](#)
- Computer Associates [177](#)
- CONTU (US National Commission on New Technological Uses of Copyrighted Works) [86-87](#)
- copy protection *see* technical protection
- copyleft *see* reciprocity
- copyright
 compensation mechanisms [65-67](#)
 Computer Associates v. Altai [90-91](#)
 derivative works [92-93](#), [123-138](#), [162](#)

- economics of [58-69](#)
- extent [91-93](#)
- first software copyrights [84](#)
- incentive theory [58-60](#)
- infringement risk [167-168](#)
- interoperability debate [87-91](#)
- liability [93](#)
- moral rights [93](#), [111-112](#), [114](#), [123](#), [162](#)
- modification *see* derivative works
- optimal limits [62-65](#)
- reform proposals [67-69](#)
- software copyright directive [89-90](#)
- Creative Commons [146](#), [154-161](#)
- Davis, Randall [126](#)
- Dedrick, Jason [193](#)
- Defense Advanced Research Projects Agency (DARPA) [31](#)
- derivative works *see* copyright
- Digital Equipment Corporation (DEC) [90](#)
- Digital Research [88](#), [196](#)
 - CP/M [88](#)
 - GEM [196](#)
- Drucker, Peter [46](#)
- dmal licensing [206-217](#)
- Electronic Frontier Foundation [43](#)
- economics
 - network approach [7-8](#)
- Emacs [33-34](#)
- Eng, Eirik [212](#)
- Fabry, Bob [31](#)
- Fershtman, Chaim [122](#)
- Fink, Martin [43](#)
- Foray, Dominique [77](#)
- Ford, Nelson [25](#)
- Forrester Research [38](#)
- Free Software Foundation [31](#), [33-34](#), [43](#), [80](#), [120](#), [129](#), [135](#), [147](#), [152](#), [167](#), [178](#), [201-203](#), [211](#), [218](#)
- Frijtsen [89](#)
- Gandel, Neil [122](#)
- Gates, Bill [40](#)
- Gnome [203](#)
- GNU/Linux *see* Linux
- GNU
 - Compiler Collection license exception [133](#)
 - Crypto license exception [148-149](#)
 - Emacs [34](#)
 - General Public License (GPL) [9](#), [34-36](#), [124-141](#), [147-150](#), [168](#), [172](#), [178](#), [180-181](#), [202](#), [211-212](#), [214](#)
 - Lesser General Public License (LGPL) [146-150](#), [180](#)
 - licenses [119-120](#), [122](#)
 - Manifesto [33](#)
 - Project [33](#), [41](#), [202](#)
- Goetz, Marty [83](#)
- Google [19](#), [138](#)
- Gosling, James [33-34](#)
- Gottinger, Hans W. [50](#)
- GPL *see* GNU General Public License
- Guile [148](#)
- hacker [35](#), [40](#), [218](#)
- Hayek, Friedrich A. [72](#)
- Hewlett-Packard (HP) [21](#), [37](#), [39](#), [104](#), [164-165](#), [174-176](#)
- Hollnith, Herman [22](#)
- Hobbs, Jordan [201](#)
- Humphrey, Watts S. [22](#)
- IBM [2](#), [14](#), [16-17](#), [20-23](#), [35](#), [37-38](#), [78](#), [89-90](#), [95](#), [104](#), [121](#), [142](#), [165](#), [176-177](#), [199](#), [203](#)
 - AIX [204](#)
 - OS/2 [196](#)
 - unbundling decision [21-24](#)
- IDC [18](#), [61](#)
- incompatibility *see* compatibility
- indemnification [145-146](#), [169-170](#)
- Intel [104](#)
- intellectual property rights (IPR)
 - see also* copyright and patents
 - and competition policy [28-80](#)
 - balancing of [105-110](#)
 - definition of [3-4](#)

- infringements as accidents [166-165](#)
 - infringement risks [173-174](#), [178-179](#), [186-187](#)
 - insurance [170-171](#)
 - law and economics of [10](#)
 - management of [3](#), [60](#), [178-179](#), [221-222](#)
 - social policy [3](#), [173-174](#), [180](#), [183-186](#), [222-224](#)
 - innovations
 - an open model [75-78](#), [163](#)
 - in software industry [69-71](#)
 - and patents [71-73](#)
 - means to appropriate [74-75](#)
 - interoperability *see* compatibility and copyright
 - Jaege, Till [117](#), [134](#)
 - Jobs, Steve [200](#)
 - Joy, Bill [30](#)
 - K Desktop Environment (KDE) [203](#), [213-214](#)
 - Katoh, Masanobu [88](#)
 - Landes, William M. [63](#), [65](#)
 - law
 - comparative law [8-9](#)
 - and economics [7-8](#), [10](#)
 - Lemley, Mark A. [74](#)
 - Leonard, Andrew [123](#)
 - Lerner, Josh [122](#), [214](#)
 - Lessig, Lawrence [155](#), [221](#)
 - Levin, Richard C. [72](#)
 - LGPL *see* GNU Lesser General Public License
 - Liebowitz, Stan [50](#), [63](#), [79](#)
 - Linix [2](#), [17-18](#), [30](#), [35-36](#), [58-59](#), [80](#), [135-136](#), [165](#), [168](#), [174-176](#), [178](#), [192-194](#), [199](#), [202-205](#), [211](#), [213](#), [216](#)
 - licensing
 - see also* open source and proprietary software
 - definition of [4](#)
 - and warranties [5](#), [123](#), [146](#), [158-159](#), [162](#), [169-170](#)
 - Locke, John [59](#)
 - Lotus [89-91](#), [101](#)
 - Mac *see* Apple
 - Machlup, Fritz [59](#), [71](#)
 - Margolis, Stephen E. [50](#), [79](#)
 - Massachusetts Institute of Technology (MIT) [33](#), [41](#)
 - MIT License [152](#)
 - Matter, Ugo [10](#)
 - McKusick, Marshall Kirk [31](#)
 - Metzger, Axel [117](#), [134](#)
 - Microsoft [1](#), [14](#), [18-20](#), [25](#), [37](#), [40](#), [47](#), [73](#), [78-80](#), [89-90](#), [95](#), [104](#), [130-131](#), [165](#), [176](#), [192-194](#), [196-201](#), [218](#)
 - and GPL [130-131](#), [198](#)
 - and open source [198](#)
 - MS/DOS [88](#), [192](#), [196](#)
 - Office [20](#)
 - Shared Source [197-198](#)
 - Windows [18-20](#), [192-193](#), [196-200](#), [204-205](#), [211](#)
- Miller, Robin [219](#)
 - MIT *see* Massachusetts Institute of Technology
 - MontaVista [203](#)
 - Mozilla [140](#)
 - Mozilla Public License [149-151](#), [180](#)
 - Mundie, Craig [1](#)
 - MySQL [2](#), [17](#), [21](#), [120](#), [134-135](#), [137](#), [140-141](#), [165](#), [206](#), [208](#), [211-212](#)
 - Netscape [1-2](#), [36](#), [149](#)
 - NewsForge [219](#)
 - NeXT [200](#)
 - NeXTStep [200](#)
 - Nichols, Kenneth [184](#)
 - Nord, Haavard [212](#)
 - North, Douglass C. [42](#), [78](#)
 - Novell [21](#), [33](#), [35](#), [90](#), [174](#), [176](#)
 - Olivetti [89](#)
 - Olsen, Michael [210](#)
 - O'Mahony, Siobhán [116](#)
 - open innovation

- Open Software License (OSL) [121](#), [138](#),
[144-146](#), [150](#), [170](#)
- open source
- academic study of [10](#)
 - benefits and challenges [38-39](#)
 - community [42-44](#), [178](#)
 - companies [172](#)
 - definition of [4](#)
 - development control [189-191](#)
 - forking [190](#), [206](#), [212](#)
 - licenses categorized [117-121](#)
 - movement *see* community
 - most popular licenses [121-122](#)
 - pricing of [188-189](#)
 - public policy initiatives [46-48](#)
- Open Source Definition [4](#), [113-116](#), [137](#),
[163](#), [188](#)
- Open Source Development Labs [175](#)
- Open Source Initiative [30](#), [36-37](#), [43-44](#),
[113](#), [121](#), [211](#), [218](#)
- history of [36-37](#)
- Open Source Risk Management Inc [168](#)
- OpenSSH [190-191](#)
- Oracle [14](#), [17](#), [37](#), [39](#), [95](#), [203](#)
- O'Reilly, Tim [36](#), [138](#)
- Pbt Consultants [184](#)
- patents
- and GPL [139-140](#)
 - and innovation [71-73](#)
 - and open source [123](#)
 - and software development [182-183](#)
 - as strategic assets [73-74](#), [121-122](#)
 - extent [99-101](#)
 - first software patents [82-83](#)
 - historical evolution [94-98](#)
 - infringement risk [168-169](#), [185-186](#)
 - international policy [98-99](#)
 - statistics [175-176](#)
 - termination clauses [180-181](#)
- Perens, Bruce [36](#)
- Perl [17](#)
- personal computer (PC) [14](#), [17](#), [24-25](#),
[104-105](#), [192-193](#)
- PHP [17](#), [131](#)
- piracy [61-62](#), [68](#)
- Plant, Arnold [61](#)
- Posner, Richard A. [63](#), [65](#), [79](#)
- Progress Software Corporation [134-135](#)
- Gemini [134-135](#)
- proprietary software
- licensing models [26-27](#), [189](#)
 - source code distribution [29-30](#)
- public domain
- Public Software Library (PsL) [25](#)
- Pugh, Emerson W. [21](#)
- Python [140](#)
- Q Public License (QPL) [213-214](#)
- Raskind, Leo [62](#)
- Ravicher, Dan [126](#)
- Raymond, Eric S. [1](#), [36](#), [42-46](#), [67](#)
- reciprocity (copyleft)
- history of [34](#)
 - definition of [117-119](#)
 - standard [146-151](#)
 - strong [124-146](#), [214](#)
- Red Hat [165](#), [174](#), [176](#)
- Riis, Thomas [62](#)
- Rosen, Lawrence E. [144-146](#)
- Samuelson, Pamela [221](#)
- SCO [35](#), [80](#), [164](#), [174-176](#)
- Seattle Software Works [88](#)
- Seltzer, Margo [209](#)
- Shapiro, Carl [74](#)
- shareware [24-25](#)
- Shy, Oz [50](#), [215](#)
- Slashdot [219](#)
- Sleepycat Software [21](#), [206](#), [208-210](#)
- BerkeleyDB (BDB) [209-210](#)
 - Sleepycat License [121](#), [209-210](#), [214](#)
- social norms [9-10](#), [116-117](#), [161](#)
- software business
- product model [20-21](#), [23](#), [26-28](#)
 - service model [20-21](#), [27-28](#)
 - definition of models [19-21](#)

- Software Choice [47](#)
- software copyright *see* copyright
- software industry
 - academic study of [11](#)
 - definition of [4](#)
 - historical overview [13-15](#)
 - innovation in [69-70](#)
 - market regions [16](#)
 - market size [15-16](#)
- software patents *see* patents
- software products
 - as capital goods [52](#)
 - as components in systems [54-55](#)
 - as information goods [51-52](#)
 - as public goods [53](#)
 - lock in [56-57](#)
 - network economics approach [50](#), [56-58](#), [190-191](#)
- Sony [104](#)
- Sourceforge [121-122](#), [214](#)
- SSH Communications Security [190-191](#)
- Stanford University [155](#)
- Stac Electronics [73](#)
- Stallman, Richard [33-34](#), [36](#), [41-46](#), [89](#), [120](#), [147](#), [178](#), [202](#), [218-219](#)
- Storage Technology [90](#)
- SUN Microsystems [95](#), [104](#), [121](#), [177](#), [203-204](#)
 - Sun Public License [121](#)
 - Sun Industry Standards Source License [121](#)
 - Solaris [177](#), [204](#)
- technical protection [22-23](#), [101-105](#)
 - anti-circumvention regulation [102-103](#)
 - trusted computing [104-105](#), [199-200](#)
- Thiése, Jacques-François [215](#)
- Tirole, Jean [122](#), [214](#)
- Torrini, Salvatore [16](#), [74](#)
- Torvalds, Linus [9](#), [35](#), [41-42](#), [44](#), [135-137](#), [180](#), [202](#)
- Trolltech [21](#), [212-214](#)
 - Qt [212-213](#)
- University of California (at Berkeley) [30-31](#), [41](#), [120](#), [152](#), [209](#)
- University of Helsinki [41](#)
- Unix [24](#), [30-33](#), [41](#), [55](#), [80](#), [192-193](#), [199](#), [201-202](#), [204](#), [211](#), [213](#)
- Unix System Laboratories [32-33](#)
- Wall, Larry [154](#)
- Watson, Thomas J. Sr. and Jr. [22](#)
- West, Joel [193](#)
- Whitlow, Duane [83](#)
- Whitlow Computer Systems [83](#)
- Widenius, Michael [211](#)
- Windows *see* Microsoft
- World Intellectual Property Organization (WIPO) [87](#), [110](#), [221](#), [223](#)
 - software model law proposal [85-86](#)
- World Trade Organization (WTO) [99](#), [223](#)
 - Treaty on Trade Related Aspects of Intellectual Property (TRIPS) [99](#)
- WordPerfect [90](#)
- X11 [203](#)
- Yahoo [138](#)
- Ylönen, Tatu [190](#)